STABILIZATION OF TIME-DELAY SYSTEMS BY NONCONVEX OPTIMIZATION TECHNIQUES

Master of Science Thesis

Süleyman Mert ÖZER

Eskişehir, 2016

STABILIZATION OF TIME-DELAY SYSTEMS BY NONCONVEX OPTIMIZATION TECHNIQUES

Süleyman Mert ÖZER

MASTER OF SCIENCE THESIS

Graduate School of Sciences
Electrical and Electronics Engineering Program
Supervisor: Prof. Dr. Altuğ İFTAR

Eskişehir
Anadolu University
Graduate School of Sciences
September, 2016

This thesis work is supported in part by the Scientific Research Projects Commission of Anadolu University under the Master Thesis grant 1410F418 and in part by the Scientific and Technical Research Council of Turkey (TÜBİTAK) under grants 112E153 and 115E379.

FINAL APPROVAL FOR THESIS

This thesis titled "Stabilization of Time-Delay Systems by Nonconvex Optimization Techniques" has been prepared and submitted by Süleyman Mert ÖZER in partial fullfillment of the requirements in "Anadolu University Directive on Graduate Education and Examination" for the Degree of Master of Science in Electrical and Electronics Engineering Department has been examined and approved on 25/08/2016.

Committe Members		**Signature**
Member (Supervisor)	: Prof. Dr. Altuğ İFTAR	.....
Member	: Prof. Dr. Arif Bülent ÖZGÜLER	.....
Member	: Assist. Prof. Dr. Hanife APAYDIN ÖZKAN	.....

....

Date Director

Graduate School of Science

ABSTRACT

STABILIZATION OF TIME-DELAY SYSTEMS
BY NONCONVEX OPTIMIZATION TECHNIQUES

Süleyman Mert ÖZER

Electrical and Electronics Engineering Program
Anadolu University, Graduate School of Sciences, September, 2016

Supervisor: Prof. Dr. Altuğ İFTAR

In this thesis, both centralized and decentralized controller design for linear time-invariant time-delay systems is considered. The main objective is to design a controller which strongly stabilizes the given time-delay system. In this respect, the centralized controller design method, which uses nonsmooth and nonconvex optimization based fixed-order controller design approach, is introduced. Regarding the nonconvexity, an initialization procedure is proposed as a novel contribution of this thesis. Then, the centralized controller design algorithm, which uses this initialization procedure and applies nonsmooth optimization algorithms, is given. After that, considering the recently developed stabilizability conditions of decentralized time-delay systems, a decentralized controller design algorithm, based on decentralized pole assignment algorithm, is proposed. Also, a new MATLAB based software package, named as DCD-TDS, is introduced. Finally, by using DCD-TDS, the design methods are applied to numerical examples.

Keywords: Time-delay systems, stabilization, decentralized control, optimization, software development.

ÖZET

ZAMAN GECİKMELİ SİSTEMLERİN
DIŞBÜKEY OLMAYAN OPTİMİZASYON TEKNİKLERİ
İLE KARARLILAŞTIRILMASI

Süleyman Mert ÖZER

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Anadolu Üniversitesi, Fen Bilimleri Enstitüsü, Eylül, 2016

Danışman: Prof. Dr. Altuğ İFTAR

Bu tezde, doğrusal zamandan bağımsız zaman gecikmekli sistemler için hem merkezi hem de merkezi olmayan denetleyici tasarımı ele alınmıştır. Asıl amaç, verilen zaman gecikmeli sistemi kuvvetli kararlılaştıracak bir denetleyici tasarlamaktır. Bu bağlamda, düzgün ve dışbükey olmayan optimizasyon tabanlı sabit boyutlu denetleyici tasarım yaklaşımını kullanan merkezi denetleyici tasarım yöntemi sunulmuştur. Dışbükey olmayışa istinaden, bu tezin özgün bir katkısı olarak bir başlatma prosedürü önerilmiştir. Sonra, başlatma prosedürünü kullanan ve düzgün olmayan optimizasyon algoritmasını uygulayan merkezi denetleyici tasarım algoritması verilmiştir. Akabinde, yakın zamanda geliştirilen merkezi olmayan zaman gecikmekli sistemlerin kararlılık koşulları göz önüne alınarak, merkezi olmayan kutup atama algoritmasına dayanan bir merkezi olmayan denetleyici tasarım algoritması önerilmiştir. Ayrıca, DCD-TDS isminde MATLAB tabanlı yeni bir yazılım paketi de tanıtılmıştır. Son olarak, tasarım metotları DCD-TDS kullanılarak nümerik örneklere uygulanmıştır.

Anahtar Kelimeler: Zaman gecikmeli sistemler, kararlılaştırma, merkezi olmayan denetim, optimizasyon, yazılım geliştirme

ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my supervisor Prof. Dr. Altuğ İftar for his continuous support and guidance. His hard work, attention to details, patience and motivation have set an great example which I hope to match some day. I couldn't have imagined having a better advisor.

I must acknowledge my dear colleague Mr. Hüseyin Ersin Erol who is the best research partner that someone can have. He was always ready to help and share his experiences. I must also acknowledge Ms. Gizem Gülmez for her valuable contributions to the software. Besides her, Ms. Büşra Şeker, Mr. Mert Yılmaz, and Mr. Anıl Çamsarı have also contributed to the software.

I am grateful to Prof. Dr. Wim Michiels for his precious comments and support. I look forward to a continuing collaboration with him in the future. I would also like to thank the rest of my thesis committee: Prof. Dr. Arif Bülent Özgüler and Assist. Prof. Dr. Hanife Apaydın Özkan for their insightful comments.

I would like to thank to Ms. Özge Ayvazoğluyüksel, Mr. Ali Can Yağan, and Mr. Oğuzkağan Alıç for being great friends and standing by me during the worst times. I would also like to thank to all my colleagues who have been helpful and supportive.

Finally, I would like to thank my family for the support they provided me through my entire life. In particular, I greatly appreciate my mother. Without her encouragement and support, I would not have finished this thesis.

02/09/2016

STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with "scientific plagiarism detection program" used by Anadolu University, and that "it does not have any plagiarism" whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.

.....

Süleyman Mert ÖZER

TABLE OF CONTENTS

LIST OF FIGURES

Page

NOTATION

$\mathbb{R}$	Real numbers
$\mathbb{C}$	Complex numbers
$\mathbb{N}$	Natural numbers with zero
$\mathbb{R}^n$	Space of n-dimensional real vectors
$\mathbb{C}^n$	Space of n-dimensional complex vectors
$\mathbb{R}^{k\times l}$	Space of $k \times l$-dimensional real matrices
$\mathbb{C}^{k\times l}$	Space of $k \times l$-dimensional complex matrices
$\mathrm{Re}(s)$	Real part of $s \in \mathbb{C}$
$\mathrm{Im}(s)$	Imaginary part of $s \in \mathbb{C}$
$\mathbb{C}_\mu^-$	$\{s \in \mathbb{C} \mid \mathrm{Re}(s) < \mu\}$ for $\mu \in \mathbb{R}$
$\mathbb{C}_\mu^+$	$\{s \in \mathbb{C} \mid \mathrm{Re}(s) > \mu\}$ for $\mu \in \mathbb{R}$
$\bar{\mathbb{C}}_\mu^+$	$\{s \in \mathbb{C} \mid \mathrm{Re}(s) \geq \mu\}$ for $\mu \in \mathbb{R}$
j	Imaginary unit, $\mathrm{j}^2 = -1$
I	Identity matrix of appropriate dimensions
I_k	$k \times k$ identity matrix
0	Zero matrix of appropriate dimensions
0_k	$k \times k$ zero matrix
$0_{k\times l}$	$k \times l$ zero matrix
$\mathrm{rank}(\Gamma)$	Rank of a matrix Γ
$\det(\Gamma)$	Determinant of a matrix Γ
$\rho(\Gamma)$	Spectral radius of a matrix Γ
Γ^T	Transpose of a matrix or vector Γ
Γ^*	Complex-conjugate transpose of a matrix or vector Γ
Γ^{-1}	Inverse of a matrix Γ
$\bar{\nu}$	$\{1, \ldots, \nu\}$ for a positive integer ν
$\mathrm{bdiag}[\cdots]$	A block diagonal matrix with blocks $\cdots$ on its diagonal

ACRONYMS

CFM	Centralized Fixed Mode
DFM	Decentralized Fixed Mode
DDE	Delay Difference Equation
DDAE	Delay Differential Algebraic Equation
FM	Fixed Mode
GS	Gradient Sampling
GUI	Graphical-User-Interface
LTI	Linear Time-Invariant
PIP	Parity Interlacing Property
TDS	Time-Delay System
TFM	Transfer Function Matrix

1. INTRODUCTION

1.1. Overview and Motivation

Many dynamic systems may involve time-delays, either inherently or due to delays in communication channels, sensors, actuators, etc. (e.g., see [1] and references therein). Considering a feedback control system, acquiring information, creating control signals and executing decisions does not occur instantaneously [2]. Neglecting these delays may lead to a poor performance or even cause a destabilizing effect in the feedback control of dynamic systems [3]. Therefore, time-delays must be taken into account in the system modeling and during the controller design process. In particular, such systems are described by *delay-differential* or *delay-differential-algebraic equations* (DDAEs) and called *time-delay systems* (TDS). As a matter of fact, it is, in general, more difficult to analyze and stabilize time-delay systems, compared to the delay-free (lumped) systems, since their states can not be represented by finitely many state variables, thus, they have infinitely many modes (characteristic roots) [4].

In the scope of this thesis, the aim is to design a finite-dimensional dynamic output feedback controller which μ-stabilizes the given time-delay system, where $\mu \in \mathbb{R}$ is a predetermined *stability boundary*, i.e., the system is *μ-stable* if the system does not have any modes in $\bar{\mathbb{C}}_{\mu}^{+}$ or any chain of modes approaching $\bar{\mathbb{C}}_{\mu}^{+}$. Here, the system to be considered may be retarded or neutral type. It is well-known that, for a retarded type system, there always exist finitely many modes in any right half plane [2]. On the other hand, a neutral system induces complications compared to a retarded system. The reason behind this situation is the existence of the associated *Delay-Difference-Equation* (DDE) which plays a central role in the analysis of a neutral type system. Because of the DDE, a neutral system exhibit chains of modes, whose imaginary parts tend to infinity, yet whose real parts have always a finite limit [5]. Thus, the μ-stability may not be determined by calculating finite number of modes in a compact set. Furthermore, the high frequency modes of these chains may be hypersensitive to delay perturbations which may yield to detract the robustness to infinitesimal delay changes. Consequently, the presence of such modes makes the determination of spectral properties (e.g., μ-stability), by

calculating finitely many modes, unreliable. This situation was the main motivation of [5] to introduce the concept of *strong stability*. More precisely, the system is said to be *strongly μ-stable* if it is μ-stable and remains μ-stable for small changes in the time-delays. In this respect, the stabilization objective is extended so that the designed controller must ensure not only the μ-stability but strong μ-stability of the closed-loop system, i.e., the closed-loop system must be robustly stable against small changes in the time-delays. However, it is not possible to strongly μ-stabilize a system, whose DDE is not strongly μ-stable, because of the fact that, in practice, any feedback controller would introduce some time-delays which are independent of the time-delays of the system itself. So, if the associated DDE is already strongly μ-stable, then, the designed controller can only maintain the strong μ-stability of the DDE of the closed-loop system. In fact, in practice, it is not possible to move the modes of the DDE by any feedback controller [6].

It should be noted that, for a retarded type system, even though the μ-stability extends to the strong μ-stability, thus the μ-stabilizing controller strongly μ-stabilizes the system at the same time, when the system has a delayed direct feedback, this situation will be dramatically changed. In that case, the closed-loop system will be neutral type. Thus, strong μ-stability must be considered, precisely. In fact, assuming that any feedback introduces some time-delays, in practice, this situation is unavoidable.

So far, many different controller design methods for time-delay systems have been proposed in the literature (e.g., see [7] and references therein). Among the existing methods, eigenvalue-based methods have become popular in the stabilization of linear time-invariant (LTI) time-delay systems [1]. First, the so-called *continuous pole placement* algorithm was presented in [8] for retarded-time delay systems, then, extended to the neutral case in [9], where strong stability context was also considered. As an alternative to the continuous pole placement method, a *nonsmooth optimization*-based *fixed-order* controller design method, which was originally introduced for robust stabilization of finite-dimensional systems in [10], was adapted to the retarded time-delay systems in [11]. This method was then employed in [12] to design strongly stabilizing state-derivative controllers for retarded time-delay systems. Although, the given system was assumed to be re-

tarded in [12], the closed-loop system becomes neutral due to the structure of the controller used. Finally, strong stabilization of neutral time-delay systems, whose autonomous parts can be described by DDAEs, was considered in [13]. In addition to these papers, fixed-order controller design method has been considered in many studies [14], [15], [16], [17]. Even though, (closed-loop) stability is not guaranteed by using this method, it can, in principle, be achieved through minimizing the real part of the rightmost mode, i.e., *spectral abscissa*, as a function of the controller parameters.

However, the main difficulty of this optimization based method is that, the objective function to be minimized is both nonsmooth and nonconvex. Because of that, a specialized algorithm must be employed, since, standard optimization algorithms may fail to overcome the nonsmoothness of the objective function [18]. For this purpose, the *gradient sampling* (GS) algorithm of [19], which is able to find local minima of general nonsmooth, nonconvex objective functions, is proposed. Alternatively, as it is shown in [20], BFGS algorithm can also be used to minimize nonsmooth objective functions. In the literature, both of them are advised to be used with an inexact line search. Furthermore, both of these methods have been coded in MATLAB and brought together in a software package named as HANSO [21].

Moreover, it has been shown in [22] for retarded systems and in [17] for neutral systems that, since the optimization problem to be solved is not convex, a completely random initialization may not always produce the best result. Therefore, special care must be given to initialize the optimization parameters, i.e., the entries of the initial controller matrices, in order to facilitate the convergence of the optimization algorithms.

In this method, the structure of the controller is also fixed (this is why, sometimes it is called as *fixed-structure* controller design). So, one possible option is to let all the elements of all the controller matrices be free parameters. However, since the input-output relation of the output feedback controller is unique only up to a similarity transformation, this option yields a unnecessarily high number of free parameters for optimization. Therefore, the controller matrices are structured in certain canonical forms in [22]. In fact, reducing the number of optimization

parameters will not only lessen the burden on the optimization algorithm, but, it will also facilitate the convergence of the algorithm by avoiding over-parametrization of the controller (e.g., see [12] for empirical evidence).

On the other hand, for many large-scale systems, it may be very costly, even impossible, to collect all the information in a centralized place, process it there, and dispatch the control commands from there [23]. For such systems, decentralized control is necessary or preferable [24]. As a result, such systems are no longer controlled by a centralized controller but by several independent controllers, each of which can observe and control only the certain part of the overall system. In this structure, these local controllers, all together, represent a decentralized controller [25].

Furthermore, many large-scale systems may involve time-delays. Recently, there have been a number of studies on this topic as well (e.g., [22], [25], [26], [27], [28] and references therein). Even though, there exist controller design methods for time-delay systems, a few of them has been adapted to decentralized case in recent years. This was the main motivation of this thesis to introduce a decentralized controller design method for time-delay systems which makes use of the (modified versions of) existing centralized controller design algorithms.

The stabilizability of a decentralized system is determined by its *fixed modes*. Basically, a mode of a LTI dynamic system is called as fixed if its location on the complex plane does not change for any LTI static output feedback controller applied to the system. In particular, a fixed-mode is called as a *decentralized fixed mode* (DFM) if the considered system is decentralized and applied controller is a decentralized controller. On the other hand, a fixed-mode is called as a *centralized fixed mode* (CFM) if the considered system is a centralized system, which, in general, may be the part of a decentralized system, and applied controller is a centralized controller.

In order to establish stabilizability conditions for a decentralized system, first, the characterization of fixed modes must be made. Luckily, there is a wide literature on this topic. The notion of fixed mode is first introduced as DFM in [29] for finite-dimensional decentralized systems. According to the notion of DFM, it has been shown that a necessary and sufficient condition for stabilizability of a

LTI decentralized finite-dimensional dynamic system by LTI decentralized finite-dimensional dynamic controllers is that it should not have any unstable DFMs. In the infinite-dimensional case, it was shown in [30] that, a LTI retarded time-delay system can be stabilized by a LTI dynamic controller if and only if the system does not have any unstable CFMs. Then, the result of [29] is extended to a LTI decentralized retarded time delay system with commensurate-time-delays in [31]. The same result is extended to retarded systems with incommensurate-time-delays in [32] and, then, to neutral systems in [33]. In this thesis, studies are carried on control of time-delay systems based on [26] which establishes the most general results on μ-stabilizability of a time-delay system.

In order to design a decentralized controller, by using the fixed-order controller design method, first attempt has been made in [22] for retarded time-delay systems. In [22], a decentralized controller design algorithm for LTI retarded time-delay systems was proposed. This algorithm is based on the *decentralized pole assignment algorithm* of [34] which was originally proposed for finite-dimensional systems. In this algorithm, a centralized controller is designed for each control agent sequentially. Therefore, a centralized controller design algorithm is needed to be used in this approach. For this purpose, nonsmooth optimization based fixed-order controller design method of [13] is used. Then, in [35], the design method of [22] is extended to the decentralized neutral time-delay systems by using the proposed method of [17] which, in particular, aims strong μ-stability.

Although, there are a few software packages to design decentralized controllers for finite-dimensional systems (e.g., [36]), there exists no such software available for decentralized time-delay systems. Therefore, a new software package, which, besides analysis, can be used to design centralized or decentralized controllers to strongly μ-stabilize time-delay systems, has been introduced in [35]. The final version of this software, named as DCD-TDS, will be presented in this thesis. It should be noted that, in the centralized controller design phase, DCD-TDS makes use of the slightly customized version of related functions in the software package TDS_STABIL [13]. On the other hand, as in TDS_STABIL, HANSO is employed to solve optimization problems. Besides the decentralized controller design ability, however, compared to TDS_STABIL, DCD-TDS have the following features:

- It allows to structure the controllers in a suitable canonical form or in any user-defined form.
- In order to choose the most favorable initial controller, it uses an initialization procedure which facilitates the convergence of the optimization algorithm.
- It has a user-friendly Graphical-User-Interface (GUI) which provides an efficient utilization.

1.2. Thesis Outline

In Chapter 2, the background for both the centralized and decentralized controller design is given. In Section 2.1, the stability of a time-delay system is outlined. In Section 2.2, first, the numerical optimization is outlined, then, the nonsmooth and nonconvex optimization algorithms of HANSO are given in the subsections. In Section 2.3, decentralized control of time-delay systems is introduced. In this section, the well-known stabilizability conditions of a decentralized time-delay system are presented with some important definitions. Also, the decentralized pole assignment algortihm of Davison and Chang [34], which is originally proposed for finite-dimensional systems, is introduced.

In Chapter 3, a centralized controller design method for centralized time-delay systems is proposed. In the beginning of this chapter, system definitions are given, first, and then, the controller design objective is stated. In Section 3.1, the structure of the controller matrices is introduced. In Section 3.2, the optimization problem is introduced. In Section 3.3, the numerical computation of the objective function and its gradient is given. In Section 3.4, the initialization procedure is proposed. In Section 3.5, the overall centralized controller design algorithm, which uses optimization algorithms of HANSO, is given.

In Chapter 4, first, decentralized time-delay system definitions are given. Then, the overall decentralized controller design algorithm, based on decentralized pole assignment algorithm and centralized controller design method given in Chapter 3, is proposed.

In Chapter 5, a MATLAB based software package, named as DCD-TDS, which can be used to analyze and stabilize a given centralized/decentralized time-

delay system, is introduced. Some important modules and related functions are presented in the consecutive sections.

In Chapter 6, the proposed controller design methods, given in Chapter 3 and Chapter 4, are illustrated by using DCD-TDS. In Section 6.1, a centralized neutral time-delay system is considered. In Section 6.2, a decentralized retarded time-delay system is considered.

In Chapter 7, the concluding remarks are given. Also, in this chapter, possible future works are suggested.

2. BACKGROUND

In this chapter, the background for the subsequent chapters is presented. In Section 2.1, the stability of a time-delay system is outlined. As it is mentioned in the previous chapter, the proposed controller design methods are based on nonsmooth optimization. Due to this, in Section 2.2, a brief introduction to the numerical optimization is given first, and then, the employed algorithms are presented in consecutive subsections. In Section 2.3, first, the background of decentralized control of time-delay systems is outlined, then, the decentralized pole assignment algorithm of Davison and Chang [34] is introduced.

2.1. Stability of Time-Delay Systems

Consider a LTI time-delay system whose autonomous part is described by the DDAEs:

$$E\dot{x}(t) = \sum_{i=0}^{\sigma} A_i x(t - h_i) \tag{2.1}$$

where $x(t) \in \mathbb{R}^n$ is the state vector at time t. The time-delays h_i $(i = 0, \ldots, \sigma)$ satisfy $h_0 < h_1 < \ldots < h_\sigma$, where σ indicates the number of distinct delays involved in (2.1). In particular, $h_0 := 0$ is used for notational convenience, i.e., $i = 0$ corresponds to the delay-free part of the system. The matrices E and A_i, $i = 0, \ldots, \sigma$, are constant real matrices. In order to ensure the solvability of (2.1), it is assumed that

$$\operatorname{rank}\left[E \;\; A_0\right] = \operatorname{rank}\left[E^T \;\; A_0^T\right]^T = n \; . \tag{2.2}$$

Definition 2.1. For any given $\epsilon \in \mathbb{R}$, the set of ϵ-modes of (2.1) is defined as

$$\Omega_\epsilon = \left\{ s \in \bar{\mathbb{C}}_\epsilon^+ \mid \det\left(\phi(s)\right) = 0 \right\} \tag{2.3}$$

where $\phi(s) := sE - \bar{A}(s)$ is the characteristic matrix of the system, where

$$\bar{A}(s) := \sum_{i=0}^{\sigma} A_i e^{-sh_i}. \tag{2.4}$$

For any given $\mu \in \mathbb{R}$, where μ is the *stability boundary*, (2.1) is said to be *μ-stable* if there exist a $\xi > 0$, such that $\Omega_{\mu-\xi} = \emptyset$. Note that, for $\mu \leq 0$, this definition is equivalent to exponential stability with a decay rate less than μ [4].

Moreover, the μ-stability condition can also be expressed in terms of the *spectral abscissa* of the system, which is defined as

$$c := \sup \{\mathrm{Re(s)} \mid \det(\phi(\mathrm{s})) = 0\} \ . \tag{2.5}$$

Then, (2.1) is μ-*stable* if and only if $c < \mu$.

The spectral characteristics of a neutral time-delay system are quite complicated than a retarded time-delay system. In the stability analysis of a neutral time-delay system, the associated delay-difference equation plays an important role. Let $\bar{n}$ denote the rank deficiency of E, i.e., $\bar{n} := n - \mathrm{rank}(E)$. Note that, when $\bar{n} = 0$ (i.e., $\mathrm{rank}(E) = n$), (2.1) describes a retarded system. In this case, the associated delay-difference equation does not exist and Ω_ϵ is a finite set for any $\epsilon \in \mathbb{R}$. In the case $1 \leq \bar{n} \leq n$, i.e., when (2.1) describes a neutral system, let $U \in \mathbb{R}^{n\times\bar{n}}$ and $V \in \mathbb{R}^{n\times\bar{n}}$ be such that

$$U^T E = 0 \quad \text{and} \quad EV = 0 \ , \tag{2.6}$$

where the columns of U and V form a minimal basis for the left and right null spaces of E, respectively. Then, due to the form of E and A_0, given in (2.2), $U^T A_0 V$ is nonsingular.

The associated DDE of (2.1) can be described as

$$\sum_{i=0}^{\sigma} \hat{A}_i \hat{x}(t - h_i) = 0 \ , \tag{2.7}$$

where $\hat{A}_i := U^T A_i V$, $i = 0, \ldots, \sigma$ and $\hat{x}(\cdot) \in \mathbb{R}^{\bar{n}}$ is a dummy state vector. Stability of (2.7) is determined by the location of the roots of its characteristic equation

$$\det(\phi_D(s)) = 0 \ , \tag{2.8}$$

where

$$\phi_D(s) := \sum_{i=0}^{\sigma} \hat{A}_i e^{-sh_i} \tag{2.9}$$

is the characteristic matrix of (2.7). In this case, (2.7) is μ-stable if and only if all the infinitely many roots of (2.8) are located to the left hand side of the $\mu - \xi$ axis, for some $\xi > 0$. We can also express this stability condition in terms of the spectral abscissa of (2.7), which is defined as,

$$c_D := \sup \{\mathrm{Re(s)} \mid \det(\phi_{\mathrm{D}}(\mathrm{s})) = 0\} \ . \tag{2.10}$$

Then, (2.7) is μ-stable if and only if $c_D < \mu$. In fact, μ-stability of (2.7) is a necessary condition for the μ-stability of (2.1) [5].

Although (2.10) is continuous in the entries of the system matrices, it is not continuous in the time-delays. As a consequence of this, the high frequency roots of the delay-difference equation, accordingly c_D, may be highly sensitive to infinitesimal perturbations in the time-delays. This situation was the main motivation of [5] to introduce the concept of *strong stability*. (2.1) is said to be *strongly μ-stable* if it is μ-stable and remains μ-stable for small changes in the time-delays. Furthermore, (2.7) is strongly μ-stable if and only if $\gamma_\mu < 1$, where, for $\sigma \geq 2$,

$$\gamma_\mu := \max_{\theta \in [0,2\pi]^{\sigma-1}} \rho\left(\hat{A}_0^{-1}\left[\hat{A}_1 e^{-\mu h_1} + \sum_{k=2}^{\sigma} \hat{A}_k e^{-\mu h_k} e^{\mathrm{j}\theta_k}\right]\right), \tag{2.11}$$

where $\theta := \{\theta_2, \ldots, \theta_\sigma\}$, and, for $\sigma = 1$,

$$\gamma_\mu := \rho\left(\hat{A}_0^{-1}\hat{A}_1 e^{-\mu h_1}\right) \tag{2.12}$$

(see, e.g., [13] for the case $\mu = 0$; the general case follows by the transformation $s \to s - \mu$). Although the above condition is enough to determine the strong μ-stability of (2.7), it is still crucial to know c_D since it gives direct information about the location of the roots of (2.8). Considering the hypersensitivity of c_D, the so-called *safe* upper bound which is robust to the perturbations in the time-delays must be defined. The safe upper bound, which will be indicated as C_D, is equal to the unique root of

$$g(\zeta) - 1 = 0, \tag{2.13}$$

where, for $\sigma \geq 2$,

$$g(\zeta) := \max_{\theta \in [0,2\pi]^{\sigma-1}} \rho\left(\hat{A}_0^{-1}\left[\hat{A}_1 e^{-\zeta h_1} + \sum_{k=2}^{\sigma} \hat{A}_k e^{-\zeta h_k} e^{\mathrm{j}\theta_k}\right]\right), \tag{2.14}$$

where $\theta := \{\theta_2, \ldots, \theta_\sigma\}$, and, for $\sigma = 1$,

$$g(\zeta) := \rho\left(\hat{A}_0^{-1}\hat{A}_1 e^{-\zeta h_1}\right). \tag{2.15}$$

Hence, the delay-difference equation (2.7) is strongly μ-stable if and only if $C_D < \mu$.

The strong μ-stability condition for (2.1), then, can be expressed in terms of c and either γ_μ or C_D. As a result, (2.1) is strongly μ-stable if and only if $c < \mu$ and $C_D < \mu$, while the latter condition can also be expressed as $\gamma_\mu < 1$.

2.2. Nonsmooth and Nonconvex Optimization

The general concern in optimization is solving the problem

$$\min_{p \in \mathbb{R}^n} f(p) \tag{2.16}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is called the *objective function*, and p is the vector of *parameters* with $n \geq 1$. By solving (2.16) iteratively, we find a $p^* \in \mathbb{R}^n$ which minimizes f. Basically, algorithms start with an initial point and generate a sequence of points $\{p_i\}_{i \in \mathbb{N}}$ that converges to a minimizer of f as $i \to \infty$.

In the field of optimization, the problems (hence, the related algorithms) are classified with respect to the characteristics of the objective function, and the strategy used to iterate. If the objective function f is convex, that is

$$f(\lambda a + (1 - \lambda)b) \leq \lambda f(a) + (1 - \lambda) f(b) \tag{2.17}$$

for all $a, b \in \mathbb{R}^n$ and all $\lambda \in [0, 1]$, then (2.16) is called as *convex optimization*. In this case, any local minimizer of f is also the global minimizer. But if f is not convex, then the algorithm seeks only the local minimizers of f without the knowledge of whether it is global or not. Furthermore, if the objective function f is smooth (i.e., continuously differentiable for all p), then (2.16) is called as *smooth optimization*. Regarding the applied strategy, most of the optimization algorithms use either *line search* or *trust region* based methods [37]. In this thesis, only line search based methods are of interest.

At i^{th} iteration of any line search method, a search direction d_i and a step length β_i, which will be taken along d_i, are obtained. Then, the next iteration can be defined as

$$p_{i+1} = p_i + \beta_i d_i. \tag{2.18}$$

Here, d_i must be a decent direction, so that, with small enough positive β_i, f can be reduced along this direction. In the general case, such a search direction is in the form

$$d_i = -{B_i}^{-1} \nabla f_i. \tag{2.19}$$

where B_i is a symmetric and nonsingular matrix, and ∇f_i indicates the gradient vector computed at p_i, assuming that f is differentiable at p_i. Note that, d_i is a

decent direction as long as B_i is positive definite. There are several line search based methods which are distinguished from each other with respect to the determination of a search direction. In the *steepest descent method*, B_i is simply the identity matrix I in an appropriate dimension. In the *Newton's method*, on the other hand, B_i is the exact Hessian $\nabla^2 f_i$. Although, the Newton's method is more powerful than the steepest descent method, obviously, it is computationally demanding. Therefore, *quasi-Newton methods* became quite popular in the optimization field, and provide an attractive alternative to Newton's method. Contrary to the Newton's method, in the quasi-Newton methods, B_i is an approximation of the Hessian which is updated after each iteration according to a certain formula [37].

In this thesis, we are interested in the optimization algorithms which are capable of solving (2.16) when f is continuous, but, in particular, nonsmooth and nonconvex function of p. The main concern in the nonsmooth optimization is to identify the minimizer by examining the subgradients which are the generalization of the concept of gradient to the nonsmooth case. Although there is a rich literature, and qualified algorithms for nonconvex optimization, a few of them are specialized for nonsmooth case. Thus, early references about nonsmooth optimization have been considered in the field of convex optimization. An excellent survey on this discussion can be found in [19]. In this respect, there are two nonsmooth optimization algorithms, which can be used for nonconvex objective functions, at the top of the literature.

First one is the *Gradient Sampling* (GS) algorithm which was first introduced in [38], then, the complete algorithm was given in [19] with a convergence analysis. Note that, the convergence analysis of [19] is improved by [39]. Thus, the GS algorithm is a specialized algorithm for nonsmooth optimization, and it has convergence guarantees when f is locally Lipschitz. However, the GS algorithm performs successfully on many nonsmooth problems where the objective function is not locally Lipschitz, even though the convergence analysis does not exist for such functions [19].

The second algorithm is one of the quasi-Newton methods, named as *BFGS* after its developers Broyden [40], Fletcher [41], Goldfarb [42], and Shanno [43]. Although, the BFGS algorithm is developed for smooth optimization, thus, the

convergence analysis does not even exist for nonsmooth optimization, it is much more efficient than GS algorithm, in practice, when a suitable inexact line search is used [44].

In fact, the GS and BFGS algorithms differ by how d_i in (2.18) is chosen, and how the local minimizer is identified. Regarding the sampling strategy, the GS algorithm is more robust, but, unfortunately, it is computationally expensive. Both of these algorithms have been coded in MATLAB and brought together in a software package named HANSO [21]. In the next two subsections, the associated algorithms of HANSO are presented, respectively. Then, the employed inexact line search method is given in Subsection 2.2.3.

2.2.1. Gradient Sampling Method

At a given iterate, the gradients of the objective function on a set of randomly generated nearby points is computed within the predetermined sampling radius ϵ. Once the bundle of gradients is collected, the descent direction is obtained by solving a quadratic program. Then, this information is used to obtain a local search direction that can be considered as an approximate ϵ-steepest descent direction.

Before starting the algorithm, let p_0 indicate the starting parameter vector. Also, fix the vector of sampling radiuses, $\epsilon_1, ..., \epsilon_{n_r}$, in a descending order where ϵ_j indicates the j^{th} radius, for $j = 1, .., n_r$, where n_r indicates the number of sampling radiuses (as a default value, $\epsilon_1 = 10^{-3}$, $\epsilon_2 = 10^{-4}$, $\epsilon_3 = 10^{-5}$). In addition to that, let k_{max} be a predetermined limit on the iteration numbers. The basic steps of the GS algorithm of HANSO are as follows:

GS Algorithm:

1) Let $j = 1$ and $k = 0$.

2) Compute $\nabla f(p_k)$, whenever f is differentiable at p_k.

3) Sample N random points in a ball with center at p_k and radius ϵ_j. Compute the additional gradients in the sampled points. Collect these gradients into a bundle, serving as an approximation for the $\partial_c f(p_k)$ (see (2.21) below).

4) Compute the vector with smallest norm out of this bundle by solving a quadratic programming. Let the search direction, d_k, be the negative of this vector. If the norm of d_k is smaller than a threshold value, continue with next step. Otherwise, go to the step 6.

5) If $j = n_r$, then stop: $f(p_k)$ corresponds to a local minimum. Let $p^* = p_k$. If $j \neq n_r$, then, let $p_0 = p_k$. Let $j = j + 1$ and $k = 0$. Go back to step 2.

6) If $\nabla f(p_k)^T d_k < 0$, i.e., d_k is a descent direction, continue with step 7. Otherwise, if $j = n_r$, then, stop: indicate $p^* = p_k$. If $j \neq n_r$, then, let $p_0 = p_k$. Let $j = j + 1$ and $k = 0$. Go back to step 2.

7) Perform a line search, along the direction d_k, to determine a step length, β_k, such that (2.23), to be given below, is satisfied. If such a β_k can not be found, and $k \neq k_{max}$, let $p_{k+1} = p_k$ and $k = k + 1$, then, go back to step 2. If such a β_k can not be found, $k = k_{max}$, and $j \neq n_r$, then, set $j = j + 1$, $p_0 = p_k$, $k = 0$ and go back to step 2. If such a β_k can not be found, $k = k_{max}$, and $j = n_r$, then, stop and indicate $p^* = p_{k_{max}}$. If β_k is found, continue with the next step.

8) Let $p_{k+1} = p_k + \beta_k d_k$. If $k \neq k_{max}$, let $k = k + 1$ and go back to step 2. If $k = k_{max}$ and $j \neq n_r$, set $j = j + 1$, $p_0 = p_k$, and $k = 0$ and go back to step 2. If $k = k_{max}$ and $j = n_r$, stop and indicate $p^* = p_{k_{max}}$.

In the above algorithm, the non-smooth steepest descent direction, d_k, is defined as the negative of the vector with smallest norm in the Clarke subdifferential at p_k, i.e.

$$d_k := -\arg \min_{x \in \partial_c f(p_k)} ||x|| \,. \tag{2.20}$$

where

$$\partial_c f(p_k) := \text{convexhull} \left\{ \lim_{p \to p_k} \frac{\partial f}{\partial p_k}(p) : p \in \mathcal{N} \right\} \tag{2.21}$$

denotes the Clarke subdifferential at p_k, which is the set containing all the Clarke subgradients at p_k, where $\mathcal{N}$ is a subset of a neighborhood around p_k. The reason of using (2.21) is to be able to detect the nonsmooth points which are usually the local

minimizers of the nonsmooth functions. Such a point is called *Clarke stationary point* if $0 \in \partial_c f(p_k)$. Thus, the convergence to the Clarke stationary point highly depends on the approximation of the Clarke subdifferential.

Note that, in the 7^{th} step of the GS algorithm, even if a proper β_k can not be found in the line search, the algorithm continues to iterate. Because, this typically means that, the gradient set is not rich enough and we should continue sampling in the ball with current radius.

2.2.2. BFGS Method

It is well-known that, in the Newton's method, the search direction is obtained as

$$d_k = -(\nabla^2 f(p_k))^{-1} \nabla f(p_k) \tag{2.22}$$

assuming that $\nabla^2 f(p_k)$ is positive definite. Obviously, the Newton method is computationally demanding. BFGS is known as the most effective quasi-Newton method that approximates the inverse Hessian matrix by using the first order gradient only. Thus, the approximation of the inverse Hessian, which is updated in each iteration with respect to the standard BFGS formula, is used to determine the search direction [37]. In [20], it is empirically shown (convergence proof is not provided) that, BFGS method with an inexact line search drives the function value to the Clarke stationary point for almost every starting point.

Before starting the algorithm, let p_0 indicate the starting parameter vector. Also, let H_k indicate the approximation of inverse Hessian matrix in the k^{th} step and set H_0 to the identity matrix in an appropriate dimension. In addition to that, let k_{max} be a predetermined limit on the iteration numbers. The basic steps of the BFGS algorithm of HANSO are as follows:

BFGS Algorithm:

1) Let $k = 0$. Compute $\nabla f(p_0)$, i.e. the gradient at p_0. If $\|\nabla f(p_0)\| < \kappa$, where κ is a predetermined tolerance, then stop and indicate $p^* = p_k$. Otherwise, continue with the next step.

2) Let $d_k = -H_k \nabla f(p_k)$. If $\nabla f(p_k)^T d_k < 0$, i.e., d_k is a descent direction, continue with the next step. Otherwise, stop and indicate $p^* = p_k$.

3) Perform a line search, along the direction d_k, to determine a step length, β_k, such that (2.23), to be given below, is satisfied. If such a β_k can not be found, then, stop and indicate $p^* = p_k$. Otherwise, let $p_{k+1} = p_k + \beta_k d_k$ and compute $\nabla f(p_{k+1})$, then, continue with the next step.

4) If $\|d_k\|\beta_k < \bar{\kappa}$, add $\nabla f(p_k)$ into a bundle. Otherwise, discard the gradients which are collected so far and let $\nabla f(p_k)$ be the only element of the new bundle. Then, compute the vector with smallest norm out of this bundle by solving a quadratic programming. If the norm of this vector is smaller than a threshold value, then, stop and indicate $p^* = p_k$. Otherwise, continue with the next step.

5) Let $y_k = \nabla f(p_{k+1}) - \nabla f(p_k)$, $V_k = I - (p_k^T y_k)^{-1} p_k y_k^T$. Update the approximation of Hessian matrix as $H_{k+1} = V_k H_k V_k^T + \beta_k (p_k^T y_k)^{-1} p_k p_k^T$. If $k = k_{max}$, then stop and indicate $p^* = p_{k_{max}}$. Otherwise let $k = k + 1$ and go back to step 2.

In [44], the BFGS method with an inexact line search is applied to the nonsmooth and nonconvex functions without doing any modification on the general BGFS method. In the 5^{th} step of the above algorithm, the approximation of the Hessian matrix is obtained with respect to the well-known *secant condition* which guarantees that H_k remains positive definite (see [37] for further details).

In the 4^{th} step of the above algorithm, considering the nonsmoothness, a simple test is provided to detect the approximate Clarke stationary points (which are, possibly, encountered as the local minimizers). The algorithm gathers predetermined number of gradients which are evaluated in consecutive points located in a small neighborhood, i.e., which is determined by $\bar{\kappa}$. Meanwhile, in each iteration, the smallest vector in the convex hull of the set, which consist of the gradients gathered so far, is obtained by solving a quadratic programming. Now, Clarke stationary points can be detected by comparing the smallest vector, which is obtained by the quadratic programming, with a predetermined tolerance on norm, e.g. κ. Notice that, this simple test is reduced to detect smooth stationary point of f whenever the set consist of a single gradient.

Actually, the reason behind the success of the BFGS method on nonsmooth functions is the ability of the inexact line search. As it is empirically shown in [20], nondifferentiable points are never encountered (besides the local minimizers), whenever a random initialization is provided. Thus, in the algorithm presented above, it is assumed that $\nabla f(p_k)$ exists at any p_k. Furthermore, the positive definiteness of the inverse Hessian is always ensured.

2.2.3. Inexact Line Search

In both of the optimization algorithms given above, a decent search direction d_k at the k^{th} iteration is determined in a certain way. Once d_k is obtained, the length of the step β_k which will be taken along the d_k must be determined. The ideal β_k which makes maximum reduction in the value of f is the global minimizer of the function $f(p_k+\beta_k d_k)$,where $\beta_k > 0$. However, solving this kind of a problem (which is called as *exact line search*) is too expensive considering the possible function and gradient evaluations. Instead, performing an *inexact line search*, which generates limited number of trial step lengths to ensure sufficient reduction in the value of f, is more practical. Therefore, in both of the above mentioned optimization methods, β_k is determined by an inexact line search which imposes the following conditions:

$$\begin{aligned} f(p_k + \beta_k d_k) &\leq f(p_k) + c_1 \beta_k \nabla f(p_k)^T p_k \\ \nabla f(p_k + \beta_k d_k)^T p_k &\geq c_2 \nabla f(p_k)^T p_k \end{aligned} \tag{2.23}$$

where the first one (called as *Armijo condition*) ensures sufficient decrease in the value of f, and the second one (called as *weak Wolfe condition*) ensures algebraic increase in the $\nabla f(p_k)$ along d_k. Here, the scalar multipliers c_1 and c_2 can be any real number satisfying $0 < c_1 < c_2 < 1$. Note that, despite of the theory, there is no harm to set c_1 to zero in practice [44]. The suitable choice of c_2 differs according to the used method. The performance of this inexact line search on nonsmooth and nonconvex functions is illustrated in several studies [44], [45]. Further discussions and the associated algorithm of HANSO can be found in [44].

2.3. Decentralized Control of Time-Delay Systems

Decentralized control systems consist of several independent control agents which can measure only a subset of all the outputs and access only a subset of all the inputs. By using these agents, it is aimed to μ-stabilize the overall closed-loop system, for a given $\mu \in \mathbb{R}$. In particular, the μ-stabilizability of a decentralized control system is determined by its fixed modes. Here, the background with necessary definitions are stated in this respect.

Consider a decentralized LTI time-delay system, to be denoted by Σ, with ν control agents, described as

$$\begin{aligned} E\dot{x}(t) &= \sum_{i=0}^{\sigma} \Big(A_i x(t-h_i) + \sum_{j=1}^{\nu} B_{j,i} u_j(t-h_i) \Big) \\ y_j(t) &= \sum_{i=0}^{\sigma} \Big(C_{j,i} x(t-h_i) + \sum_{k=1}^{\nu} D_{k,j,i} u_k(t-h_i) \Big) \ , \ j \in \bar{\nu} \ , \end{aligned} \tag{2.24}$$

where t is the time variable, $x(t) \in \mathbb{R}^n$ is the state vector at time t, and $u_j(t) \in \mathbb{R}^m$ and $y_j(t) \in \mathbb{R}^q$ are, respectively, the input and the output vectors at time t, accessible by the j^{th} control agent. Here, $h_0 := 0$ is used for notational convenience (i.e., $i = 0$ corresponds to the delay-free part of the system), σ indicates the number of distinct time-delays involved, and, for $i = 1, \ldots, \sigma$, $h_i > 0$ are the time-delays. The matrices E, A_i, $B_{j,i}$, $C_{j,i}$ and $D_{k,j,i}$, $(i = 0, \ldots, \sigma, j \in \bar{\nu}, k \in \bar{\nu})$ are constant real matrices.

It is worth to emphasize that, a decentralized neutral time-delay system which is described as

$$\begin{aligned} \dot{\tilde{x}}(t) + \sum_{i=1}^{\sigma} \big(\tilde{E}_i \dot{\tilde{x}}(t-h_i) \big) &= \sum_{i=0}^{\sigma} \big(\tilde{A}_i \tilde{x}(t-h_i) + \sum_{j=1}^{\nu} \tilde{B}_{j,i} u_j(t-h_i) \big) \\ y_j(t) &= \sum_{i=0}^{\sigma} \Big(\tilde{C}_{j,i} \tilde{x}(t-h_i) + \sum_{k=1}^{\nu} \tilde{D}_{k,j,i} u_k(t-h_i) \Big) \ , \ j \in \bar{\nu} \ , \end{aligned} \tag{2.25}$$

can be brought into the form of (2.24) by defining $\delta(t) := \tilde{x}(t) + \sum_{i=1}^{\sigma} \big(\tilde{E}_i \tilde{x}(t-h_i) \big)$ and $x(t) := \begin{bmatrix} \delta(t)^T & \tilde{x}(t)^T \end{bmatrix}^T$.

Note that, (2.1) and the autonomous part of Σ are in the same form. Therefore, strong μ-stability of Σ can be analyzed as for (2.1). Thus, the set of ϵ-modes, the μ-stability, c, c_D, C_D, and γ_μ can be defined as in Section 2.1. Although,

here, the μ-stabilizability of Σ is analyzed through its fixed modes, it is assumed that $c_D(\Sigma) < \mu$.

A controller $\mathcal{K}$ is said to *μ-stabilize* the system Σ if the closed-loop system, obtained by applying $\mathcal{K}$ to the system Σ, is μ-stable. In decentralized control system design, the main objective is to design ν decentralized controllers which μ-stabilize the system Σ. For this purpose, define three different classes of finite-dimensional controllers as follows:

- $\mathbf{K_s^c}$: *the class of centralized static LTI controllers* is all the controllers of the form:

$$u(t) = Ky(t) \ , \tag{2.26}$$

where

$$\begin{aligned} u(t) &:= \left[\begin{array}{ccc} u_1^T(t) & \cdots & u_\nu^T(t) \end{array} \right]^T \in \mathbb{R}^m \ , \\ y(t) &:= \left[\begin{array}{ccc} y_1^T(t) & \cdots & y_\nu^T(t) \end{array} \right]^T \in \mathbb{R}^q \ , \end{aligned} \tag{2.27}$$

where $m := \sum_{j=1}^{\nu} m_j$ and $q := \sum_{j=1}^{\nu} q_j$, and $K \in \mathbb{R}^{m \times q}$ is such that

$$\det \left(\lim_{\mathrm{Re}(s) \to +\infty} \phi_{\Sigma,\mathcal{K}}(s) \right) \neq 0 \ , \tag{2.28}$$

where $\phi_{\Sigma,\mathcal{K}}$ is the characteristic matrix of the closed-loop system obtained by applying controller $\mathcal{K}$ to the system Σ.

- $\mathbf{K_s^d}$: *the class of decentralized static LTI controllers* is all the controllers of the form:

$$u_j(t) = K_j y_j(t) \ , \quad j \in \bar{\nu} \ , \tag{2.29}$$

where $K_j \in \mathbb{R}^{m_j \times q_j}$, $j \in \bar{\nu}$, such that K_j satisfies (2.28).

- $\mathbf{K_d^d}$: *the class of decentralized finite-dimensional dynamic LTI controllers* is all the controllers of the form:

$$\begin{aligned} \dot{z}_j(t) &= F_j z_j(t) + G_j y_j(t) \\ u_j(t) &= H_j z_j(t) + K_j y_j(t) \end{aligned} \ , \quad j \in \bar{\nu} \ , \tag{2.30}$$

where $z_j \in \mathbb{R}^{l_j}$ is the state of the j^{th} local controller and F_j, G_j, H_j, and K_j are real constant matrices such that K_j satisfies (2.28). Here, the dimension of the controller, $l_j \in \mathbb{N}$, is arbitrary. Note that when $l_j = 0$, for all $j \in \bar{\nu}$, such a controller reduces to a decentralized static LTI controller; thus, $\mathbf{K_s^d} \subset \mathbf{K_d^d}$.

Definition 2.2. For a given $\epsilon \in \mathbb{R}$, the set of *ϵ-fixed modes* (ϵ-FMs) of Σ with respect to the class of controllers $\mathbf{K}$ is defined as

$$\Lambda_\epsilon(\Sigma, \mathbf{K}) = \{s \in \bar{\mathbb{C}}_\epsilon^+ \mid \det(\phi_{\Sigma,\mathcal{K}}(s)) = 0,\ \forall \mathcal{K} \in \mathbf{K}\} \tag{2.31}$$

where $\phi_{\Sigma,\mathcal{K}}$ is the characteristic matrix of the closed-loop system obtained by applying controller $\mathcal{K}$ to the system Σ.

In particular, when the class concerned is $\mathbf{K_s^c}$, $\Lambda_\epsilon^c(\Sigma) := \Lambda_\epsilon(\Sigma, \mathbf{K_s^c})$ indicates the set of ϵ- *centralized fixed modes* (ϵ-CFMs). On the other hand, when the class concerned is $\mathbf{K_s^d}$, $\Lambda_\epsilon^d(\Sigma) := \Lambda_\epsilon(\Sigma, \mathbf{K_s^d})$ indicates the set of ϵ- *decentralized fixed modes* (ϵ-DFMs). Note that, as it is stated in [26], $\Lambda_\epsilon^d(\Sigma)$ is exactly same for both $\mathbf{K_s^d}$ and $\mathbf{K_d^d}$. Relying on this fact, it is more practical to consider $\Lambda_\epsilon^d(\Sigma)$ for $\mathbf{K_s^d}$.

By definition, $\Lambda_\epsilon^c(\Sigma) \subset \Lambda_\epsilon^d(\Sigma) \subset \Omega_\epsilon(\Sigma)$. Accordingly, provided that $c_D(\Sigma) < \epsilon$, both $\Lambda_\epsilon^c(\Sigma)$ and $\Lambda_\epsilon^d(\Sigma)$ are finite sets. Therefore, the elements of these sets can be computed by either a *numerical procedure* or a *rank test* given in [26] (see Section 5.4).

In particular, the ϵ-DFMs of Σ can be determined by using the following result of [26].

Lemma 1: Let $\text{Re}(s_0) \geq \epsilon$. $s_0 \in \Lambda_\epsilon^d(\Sigma)$ if and only if there exists $k \in \{0, \ldots, \nu\}$ and $\{i_1, \ldots, i_k\} \subset \bar{\nu}$ where $i_1, \ldots, i_k$ are distinct, such that

$$\text{rank} \begin{bmatrix} \bar{A}(s_0) - s_0 E & \bar{B}_{i_1}(s_0) & \ldots & \bar{B}_{i_k}(s_0) \\ \bar{C}_{i_{k+1}}(s_0) & \bar{D}_{i_{k+1},i_1}(s_0) & \ldots & \bar{D}_{i_{k+1},i_k}(s_0) \\ \vdots & \vdots & \ddots & \vdots \\ \bar{C}_{i_\nu}(s_0) & \bar{D}_{i_\nu,i_1}(s_0) & \ldots & \bar{D}_{i_\nu,i_k}(s_0) \end{bmatrix} < n \tag{2.32}$$

where $\{i_{k+1}, \ldots, i_\nu\} := \bar{\nu} \setminus \{i_1, \ldots, i_k\}$,

$$\bar{A}(s) := \sum_{i=0}^{\sigma} A_i e^{-s h_i} \quad , \tag{2.33}$$

and, for $j, k \in \bar{\nu}$,

$$\bar{B}_j(s) := \sum_{i=0}^{\sigma} B_{j,i} e^{-sh_i} \ , \quad \bar{C}_j(s) := \sum_{i=0}^{\sigma} C_{j,i} e^{-sh_i} \ , \tag{2.34}$$

$$\bar{D}_{j,k}(s) := \sum_{i=0}^{\sigma} D_{j,k,i} e^{-sh_i} \ . \tag{2.35}$$

Since, $\Lambda_\epsilon^d(\Sigma) \subset \Omega_\epsilon(\Sigma)$, the test in Lemma 1 needs to be carried out only for $s_0 \in \Omega_\epsilon(\Sigma)$, which is a finite set for any $\epsilon > c_D$.

From Lemma 2 in [26], provided that $\mu > c_D(\Sigma^c)$, where Σ^c is a centralized system (i.e., the system in the form of (2.24) when $\nu = 1$), there exist a centralized controller (e.g., controller $\mathcal{K} \in \mathbf{K_d}$ with $\nu = 1$) which μ-stabilizes the system Σ^c if and only if $\Lambda_\mu^c(\Sigma^c) = \emptyset$.

It was further shown in [26], provided that $c_D(\Sigma) < \mu$, Σ can be μ-stabilized by a controller in the class $\mathbf{K_d}$ if and only if $\Lambda_\mu^d(\Sigma) = \emptyset$. From this follows, to determine whether or not it is possible to μ-stabilize Σ, it suffices to compute $\Lambda_\epsilon^d(\Sigma)$, for some $\epsilon < \mu$.

There are several decentralized controller synthesis algorithms developed in the last four decades [25]. Relying on Theorem 3 in [34], which basically states that there exist a decentralized LTI controller which can stabilize the system if and only if the system does not have any unstable DFM, Davison and Chang proposed the decentralized pole assignment algorithm for finite-dimensional decentralized systems. In the decentralized pole assignment algorithm, a centralized controller is designed first by using only one of the control agents to eliminate as many unstable modes as possible, except for the unstable CFM(s) of the associated centralized system. After this controller is designed, the corresponding loop is closed and a decentralized control system with $\nu - 1$ control agents is obtained. After repeating this procedure ν times, the overall decentralized controller is obtained.

Then, relying on Lemma 3 in [26], this algorithm is extended to the design of decentralized controllers in the form of (2.30) for retarded and neutral decentralized time-delay systems, respectively in [22] and in [35]. The related algorithm is given in Chapter 4.

3. CENTRALIZED CONTROLLER DESIGN

Let us consider a centralized LTI time-delay system Σ^c, described by delay-differential-algebraic equations:

$$\begin{aligned} E\dot{x}(t) &= \sum_{i=0}^{\sigma} \big(A_i x(t-h_i) + B_i u(t-h_i)\big) \\ y(t) &= \sum_{i=0}^{\sigma} \big(C_i x(t-h_i) + D_i u(t-h_i)\big) \end{aligned} \tag{3.1}$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, and $y(t) \in \mathbb{R}^q$ are, respectively, the state, the input, and the output vectors at time t. $h_1, \ldots, , h_\sigma > 0$ are the time-delays, where σ is the number of distinct time-delays of the system. $h_0 := 0$ is used for notational convenience. The matrices E, A_i, B_i, C_i, and D_i, $i = 0, \ldots, \sigma$, are constant real matrices.

The system description (3.1) is quite general as illustrated in [14]. Furthermore, it is worth to emphasize that, a neutral time-delay system which is described as

$$\begin{aligned} \dot{\tilde{x}}(t) + \sum_{i=1}^{\sigma} \big(\tilde{E}_i \dot{\tilde{x}}(t-h_i)\big) &= \sum_{i=0}^{\sigma} \big(\tilde{A}_i \tilde{x}(t-h_i) + \tilde{B}_i u(t-h_i)\big) \\ y(t) &= \sum_{i=0}^{\sigma} \big(\tilde{C}_i \tilde{x}(t-h_i) + \tilde{D}_i u(t-h_i)\big) \end{aligned} \tag{3.2}$$

can be brought into the form of (2.24) by defining $\delta(t) := \tilde{x}(t) + \sum_{i=1}^{\sigma} \big(\tilde{E}_i \tilde{x}(t-h_i)\big)$ and $x(t) := \begin{bmatrix} \delta(t)^T & \tilde{x}(t)^T \end{bmatrix}^T$.

In order to strongly μ-stabilize Σ^c, we consider finite-dimensional LTI output feedback controllers of the form

$$\begin{aligned} \dot{z}(t) &= Fz(t) + Gy(t) \\ u(t) &= Hz(t) + Ky(t) \end{aligned}, \tag{3.3}$$

where $z(t) \in \mathbb{R}^l$ is the state of the controller at time t and F, G, H, and K are real constant matrices. Here, the dimension of the controller, $l \in \mathbb{N}$, is arbitrary. Notice that when $l = 0$, (3.3) describes a static controller in the form of (2.26).

When the controllers of the form (3.3) are used, one possible option is to let all the elements of all the controller matrices be free parameters. However, since the input-output relation of the controller (3.3) is unique only up to a similarity

transformation, this option leads to a unnecessarily high number of free parameters for optimization. Therefore, we structure the controllers in certain canonical forms (see Section 3.1). In any one of these forms, the number of free parameters is equal to

$$\tilde{l} := l(q+m) + qm \; . \tag{3.4}$$

Therefore, compared to the case when all the entries of all the controller matrices are taken as free parameters (in which case the number of free parameters is $l^2 + lq + lm + qm$), the number of free parameters are reduced by l^2.

Considering (3.1) and (3.3), the closed-loop system, which will be indicated by Σ^{cl}, can be described as

$$\mathcal{E}\dot{\eta}(t) = \mathcal{A}_0(p)\eta(t) + \sum_{i=1}^{\sigma} \mathcal{A}_i \eta(t-h_i) \tag{3.5}$$

where $\eta(t) := \begin{bmatrix} x(t)^T & y(t)^T & z(t)^T & u(t)^T \end{bmatrix}^T \in \mathbb{R}^{\tilde{n}}$ is the new state vector at time t, where $\tilde{n} := n+m+q+l$,

$$\mathcal{E} := \begin{bmatrix} E & 0 & 0 & 0 \\ 0 & 0_{q\times q} & 0 & 0 \\ 0 & 0 & I_l & 0 \\ 0 & 0 & 0 & 0_{m\times m} \end{bmatrix}, \quad \mathcal{A}_0(p) := \begin{bmatrix} A_0 & 0 & 0 & B_0 \\ C_0 & -I_q & 0 & D_0 \\ 0 & G & F & 0 \\ 0 & K & H & -I_m \end{bmatrix},$$

and, for $i = 1, \ldots, \sigma$,

$$\mathcal{A}_i := \begin{bmatrix} A_i & 0 & 0 & B_i \\ C_i & 0_{q\times q} & 0 & D_i \\ 0 & 0 & 0_{l\times l} & 0 \\ 0 & 0 & 0 & 0_{m\times m} \end{bmatrix} .$$

Here, $p \in \mathbb{R}^{\tilde{l}}$ denotes the vector of free controller parameters, where $\tilde{l}$ is given in (3.4). Notice that only $\mathcal{A}_0$ depends on p, since the controller matrices appear only in $\mathcal{A}_0$.

Furthermore, if $\operatorname{rank}(E) = n$ (i.e., if Σ^c is a retarded system), let

$$\mathcal{U} = \mathcal{V} := \begin{bmatrix} 0_{n\times q} & 0 \\ I_q & 0 \\ 0_{l\times q} & 0 \\ 0 & I_m \end{bmatrix}, \tag{3.6}$$

otherwise,

$$\mathcal{U} := \begin{bmatrix} U & 0 & 0 \\ 0 & I_q & 0 \\ 0 & 0_{l\times q} & 0 \\ 0 & 0 & I_m \end{bmatrix}, \quad \mathcal{V} := \begin{bmatrix} V & 0 & 0 \\ 0 & I_q & 0 \\ 0 & 0_{l\times q} & 0 \\ 0 & 0 & I_m \end{bmatrix} \tag{3.7}$$

where U and V are as in (2.6). Also let $\tilde{\mathcal{U}} \in \mathbb{R}^{\tilde{n}\times(n-\bar{n}+l)}$ and $\tilde{\mathcal{V}} \in \mathbb{R}^{\tilde{n}\times(n-\bar{n}+l)}$ be such that

$$\bar{\mathcal{U}} := \begin{bmatrix} \tilde{\mathcal{U}} & \mathcal{U} \end{bmatrix} \quad \text{and} \quad \bar{\mathcal{V}} := \begin{bmatrix} \tilde{\mathcal{V}} & \mathcal{V} \end{bmatrix}$$

are nonsingular. Note that, the closed-loop characteristic matrix is $\phi_{\Sigma^{cl}}(s) := s\mathcal{E} - \bar{\mathcal{A}}(s)$, where

$$\bar{\mathcal{A}}(s) := \sum_{i=0}^{\sigma} \mathcal{A}_i e^{-sh_i} \ . \tag{3.8}$$

Then, the closed-loop system is well-posed, i.e., (2.28) is satisfied, if and only if

$$\det\left(\lim_{\mathrm{Re}(s)\to+\infty}\left(s\mathcal{E} - \bar{\mathcal{A}}(s)\right)\right) = \det\left(\lim_{\mathrm{Re}(s)\to+\infty}\bar{\mathcal{U}}^T\left(s\mathcal{E} - \bar{\mathcal{A}}(s)\right)\bar{\mathcal{V}}\right) \neq 0 \tag{3.9}$$

Noting that $\lim_{\mathrm{Re}(s)\to+\infty} \bar{\mathcal{A}}(s) = \mathcal{A}_0$, this reduces to

$$\det\left(\begin{bmatrix} U^T A_0 V & 0 & U^T B_0 \\ C_0 V & -I_q & D_0 \\ 0 & K & -I_m \end{bmatrix}\right) \neq 0 \ . \tag{3.10}$$

Recalling that $U^T A_0 V$ is nonsingular, this further reduces to

$$\det\left(I - K\left(D_0 - C_0 V (U^T A_0 V)^{-1} U^T B_0\right)\right) \neq 0 \ . \tag{3.11}$$

Note that, when Σ^c is retarded, (3.11) reduces to

$$\det\left(I - K D_0\right) \neq 0 \ . \tag{3.12}$$

Therefore, to guarantee the well-posed of the closed-loop system, K must be chosen to satisfy (3.11) (or (3.12) when Σ^c is retarded).

On the other hand, the associated delay-difference equation of (3.5) is

$$\mathcal{H}_0(p)\hat{\eta}(t) + \sum_{i=1}^{\sigma} \mathcal{H}_i \hat{\eta}(t-h_i) = 0 \ , \tag{3.13}$$

where $\hat{\eta}(\cdot) \in \mathbb{R}^{\hat{n}}$ is a dummy state vector, where $\hat{n} := \bar{n} + m + q$, where $\bar{n} := n - \text{rank}(E)$. Also, $\mathcal{H}_0(p) := \mathcal{U}^T \mathcal{A}_0(p) \mathcal{V}$, which depends only on the K matrix of the controller, and, for $i = 1, \ldots, \sigma$, $\mathcal{H}_i := \mathcal{U}^T \mathcal{A}_i \mathcal{V}$, which are independent of the controller.

Note that, Σ^{cl} and the autonomous part of Σ^c are both in the form of (2.1). Therefore, strong μ-stability of both the open-loop and closed-loop system can be analyzed as for (2.1). Thus, the set of ϵ-modes, the strong μ-stability, c, C_D, and γ_μ can be defined as in Section 2.1.

In particular, Σ^{cl} is strongly μ-stable if and only if $c(\Sigma^{cl}) < \mu$ and $C_D(\Sigma^{cl}) < \mu$, while the latter condition can also be expressed as $\gamma_\mu(\Sigma^{cl}) < 1$. Since these quantities depend on the free parameter vector, $p \in \mathbb{R}^{\tilde{l}}$, of the controller, from here on, for a particular $p \in \mathbb{R}^{\tilde{l}}$, we will denote $c(\Sigma^{cl})$, $C_D(\Sigma^{cl})$, and $\gamma_\mu(\Sigma^{cl})$ as $c(p)$, $C_D(p)$, and $\gamma_\mu(p)$, respectively. We note, however, that, although $c(p)$ depends on the complete $p \in \mathbb{R}^{\tilde{l}}$, in general, $C_D(p)$ and $\gamma_\mu(p)$ depend only on the K matrix, since (3.13) depends only on the K matrix. Therefore, from hereafter, $C_D(p)$ and $\gamma_\mu(p)$ will be denoted as $C_D(p^K)$ and $\gamma_\mu(p^K)$, exclusively, where p^K is the part of p that consist of the elements of K. Also note that $\gamma_\mu(p^K)$ is finite for any $\mu \in \mathbb{R}$ if and only if (3.11) (or (3.12) when Σ^c is retarded) is satisfied. Therefore, keeping $\gamma_\psi(p^K) < 1$, for any $\psi \in \mathbb{R}$, also guarantees the well-posedness of the closed-loop system.

It is worth to note that, even if the given system Σ^c is a retarded system, Σ^{cl} would in general be a neutral system when $D_i \neq 0$, for at least one $i \in \{1, \ldots, \sigma\}$. Furthermore, this result can also be generalized to the case of $D_0 \neq 0$, since any feedback would introduce a delay in practice [46].

Hence, the aim is to design a controller of the form (3.3), i.e., to find an $l \in \mathbb{N}$ and a $p \in \mathbb{R}^{\tilde{l}}$, which makes Σ^{cl} strongly μ-stable. Here, $\tilde{l}$ depends on l through (3.4). It is important to note that, since in practice any feedback controller would introduce some time-delays, no matter how small, which are independent of the time-delays of the given system, it is not possible to reduce $C_D(\Sigma)$ by feedback. Because of this reason, hereafter, it is assumed that Σ^c satisfies $C_D(\Sigma^c) < \mu$, i.e., $\gamma_\mu(\Sigma^c) < 1$. Furthermore, it is also assumed that $\Lambda_\mu^c(\Sigma^c) = \emptyset$.

To achieve our aim, stated above, a nonsmooth optimization based method

is proposed. Before getting into the related optimization problem, the structure of the controller matrices is introduced in the next section. Then, the optimization problem is given in Section 3.2. In Section 3.3, the evaluation of the corresponding objective function and its gradient is given. Then, the initialization procedure is introduced in Section 3.4. Finally, the overall centralized controller design algorithm is given in Section 3.5.

3.1. Structure of the Controller Matrices

Let the dimension of the output, y, be q and the dimension of the input, u, be m. Then the controller has q inputs and m outputs. If $q \leq m$, then the *multivariable controllable canonical form* is used for the controller. If the dimension of the controller, l, is greater than or equal to q, then the controller matrices take the following form:

$$F = \begin{bmatrix} 0_{(l-q)\times q} & & I_{l-q} \\ a_{1,1} & \dots & a_{1,l} \\ \vdots & & \vdots \\ a_{q,1} & \dots & a_{q,l} \end{bmatrix}, \quad G = \begin{bmatrix} 0_{(l-q)\times q} \\ I_q \end{bmatrix},$$
$$H = \begin{bmatrix} c_{1,1} & \dots & c_{1,l} \\ \vdots & & \vdots \\ c_{m,1} & \dots & c_{m,l} \end{bmatrix}, \quad K = \begin{bmatrix} d_{1,1} & \dots & d_{1,q} \\ \vdots & & \vdots \\ d_{m,1} & \dots & d_{m,q} \end{bmatrix}. \tag{3.14}$$

where $a_{i,j}$ $(i = 1, \dots, q,\ j = 1, \dots, l)$, $c_{i,j}$ $(i = 1, \dots, m,\ j = 1, \dots, l)$, and $d_{i,j}$ $(i = 1, \dots, m,\ j = 1, \dots, q)$ are the free parameters. If $l < q$, then the controller matrices take the following form:

$$F = \begin{bmatrix} a_{1,1} & \dots & a_{1,l} \\ \vdots & & \vdots \\ a_{l,1} & \dots & a_{l,l} \end{bmatrix}, \quad G = \begin{bmatrix} & b_{1,1} & \dots & b_{1,q-l} \\ I_l & \vdots & & \vdots \\ & b_{l,1} & \dots & b_{l,q-l} \end{bmatrix},$$
$$H = \begin{bmatrix} c_{1,1} & \dots & c_{1,l} \\ \vdots & & \vdots \\ c_{m,1} & \dots & c_{m,l} \end{bmatrix}, \quad K = \begin{bmatrix} d_{1,1} & \dots & d_{1,q} \\ \vdots & & \vdots \\ d_{m,1} & \dots & d_{m,q} \end{bmatrix}, \tag{3.15}$$

where, $a_{i,j}$ $(i = 1, \dots, l,\ j = 1, \dots, l)$, $c_{i,j}$ $(i = 1, \dots, m,\ j = 1, \dots, l)$, $d_{i,j}$ $(i = 1, \dots, m, j = 1, \dots, q)$, and $b_{i,j}$ $(i = 1, \dots, l, j = 1, \dots, q-l)$ are the free parameters.

If $q > m$, then the *multivariable observable canonical form* is used for the controller. If $l \geq m$ then the controller matrices take the following form:

$$F = \begin{bmatrix} 0_{m\times(l-m)} & a_{1,1} & \dots & a_{1,m} \\ & \vdots & & \vdots \\ I_{l-m} & a_{l,1} & \dots & a_{l,m} \end{bmatrix}, \quad G = \begin{bmatrix} b_{1,1} & \dots & b_{1,q} \\ \vdots & & \vdots \\ b_{l,1} & \dots & b_{l,q} \end{bmatrix}$$
$$H = \begin{bmatrix} 0_{m\times(l-m)} & I_m \end{bmatrix}, \quad K = \begin{bmatrix} d_{1,1} & \dots & d_{1,q} \\ \vdots & & \vdots \\ d_{m,1} & \dots & d_{m,q} \end{bmatrix} \tag{3.16}$$

where $a_{i,j}$ $(i = 1, \dots, l,\ j = 1, \dots, m)$, $b_{i,j}$ $(i = 1, \dots, l,\ j = 1, \dots, q)$, and $d_{i,j}$ $(i = 1, \dots, m,\ j = 1, \dots, q)$ are the free parameters. If $l < m$ then the controller matrices take the following form:

$$F = \begin{bmatrix} a_{1,1} & \dots & a_{1,l} \\ \vdots & & \vdots \\ a_{l,1} & \dots & a_{l,l} \end{bmatrix}, \quad G = \begin{bmatrix} b_{1,1} & \dots & b_{1,q} \\ \vdots & & \vdots \\ b_{l,1} & \dots & b_{l,q} \end{bmatrix},$$
$$H = \begin{bmatrix} & I_l & \\ c_{1,1} & \dots & c_{1,l} \\ \vdots & & \vdots \\ c_{m-l,1} & \dots & c_{m-l,l} \end{bmatrix}, \quad K = \begin{bmatrix} d_{1,1} & \dots & d_{1,q} \\ \vdots & & \vdots \\ d_{m,1} & \dots & d_{m,q} \end{bmatrix} \tag{3.17}$$

where, $a_{i,j}$ $(i = 1, \dots, l,\ j = 1, \dots, l)$, $c_{i,j}$ $(i = 1, \dots, m-l,\ j = 1, \dots, l)$, $d_{i,j}$ $(i = 1, \dots, m,\ j = 1, \dots, q)$, and $b_{i,j}$ $(i = 1, \dots, l,\ j = 1, \dots, q)$ are the free parameters.

Note that, in all the forms above, the number of free parameters is $\tilde{l}$, as given in (3.4).

3.2. Optimization Problem

As it is stated in Section 2.1, Σ^{cl} is strongly μ-stable if and only if $c(\Sigma^{cl}) < \mu$ and $\gamma_\mu(\Sigma^{cl}) < 1$. The proposed method is based on minimizing $c(p)$ over p such that, for some $\psi < \mu$, $\gamma_\psi(p^K) < 1$. Here, $\psi \in \mathbb{R}$ is the *strong stability boundary* introduced for the infinite chains of modes. Since it is not possible, in practice, to reduce $C_D(\Sigma^c)$, one should choose $\psi > C_D(\Sigma^c)$. This constrained optimization problem can be reformulated as an unconstrained optimization problem, as in [12], by using

the so-called *barrier method*. By this way, the optimization problem becomes

$$\min_{p} f(p) \tag{3.18}$$

where

$$f(p) := c(p) - r \log(1 - \gamma_\psi(p^K)) \; . \tag{3.19}$$

Here, $r > 0$ is a weighting parameter and the logarithmic term is the barrier function, which is used to penalize $c(p)$ for violations of the constraint $\gamma_\psi(p^K) < 1$. This is achieved since $f(p)$ converges to infinity as $\gamma_\psi(p^K)$ approaches 1. Thus, if a local minimizer, p^*, of (3.18) corresponds to $c(p^*) < \mu$, then the controller corresponding to p^* strongly μ-stabilizes Σ^c.

Notice that, if Σ^c is a retarded system with $D_i = 0$, $i = 1, ..., \sigma$, strong μ-stability can be reduced to μ-stability only, thus, it is more practical to let $f(p) := c(p)$.

As indicated in [11], $c(p)$ is a non-convex function of p. Therefore, there might be several local minima. Furthermore, $c(p)$ is nonsmooth when it is determined by more than one modes of (3.5) which have same real parts. However, for almost every p, gradient of $c(p)$ exists [12]. Furthermore, for almost every p, gradient of $\gamma_\psi(p^K)$ exists, thus of $f(p)$. Relying on that, the optimization problem (3.18) can be solved by using a gradient based algorithm. However, standard gradient based algorithms may fail because of the nonsmoothness of the objective function. Therefore, a specialized algorithm must be employed. In this respect, the GS algorithm of [19] and the BFGS method of [44] is used to solve (3.18). The details of the optimization algorithms have already been given in Section 2.2. Furthermore, the associated software package of these algorithms, named as HANSO, is embedded to the developed software package.

The optimization algorithms of HANSO do not require any information about the structure of the objective function f (such as subgradient or subdifferential formulas, partial smoothness of the function). Instead, a user-provided routine, which returns the function value $f(p)$ and the gradient vector $\nabla f(p)$ at p whenever f is differentiable at p, is needed. Indeed, considering the round-off errors, it is impractical to determine whether f is differentiable at p or not in the sense of numerical computation. The computation of $f(p)$ and $\nabla f(p)$ are given in the next

section. Moreover, HANSO requires an initial parameter vector as a starting point. In this respect, the developed initialization procedure is given in Section 3.4.

3.3. Evaluations of the Objective Function and Gradients

The first step in the computation of $f(p)$ is the evaluation of $\gamma_\psi(p^K)$ which can be handled as given in Section 5.6. Then, the spectral abscissa, $c(p)$, can be calculated by computing the ϵ-modes of Σ^{cl}, for $\psi < \epsilon$, as described in Section 5.3. Once the set of ϵ-modes are computed, the spectral abscissa can be determined as the real part of the rightmost ϵ-mode.

Accordingly, the partial derivative of $f(p)$ w.r.t each controller parameter can be calculated as,

$$\frac{\partial c}{\partial p_k}(p) + \frac{r}{1-\gamma_\psi(p^K)}\left(\frac{\partial \gamma_\psi}{\partial p_k}(p^K)\right). \tag{3.20}$$

Firstly, the partial derivatives of $c(p)$ can be obtained as the sensitivity of the rightmost mode w.r.t the free parameters of the controller. If there exists multiple of such (simple) modes, then, one of these modes is chosen. For, $k = 1, ..., \tilde{l}$, it can be expressed as

$$\frac{\partial c}{\partial p_k}(p) = \mathrm{Re}\left(\frac{w^*\left(\frac{\partial \mathcal{A}_0}{\partial p_k}(p)\right) z}{w^*\left(\mathcal{E} + \sum_{i=1}^{\sigma} \mathcal{A}_i h_i e^{-s_o h_i}\right) z}\right) \tag{3.21}$$

where $\mathcal{E}$, $\mathcal{A}_0(p)$, and $\mathcal{A}_i$, $i = 1, \ldots, \sigma$, are the matrices of the closed-loop system written in the form (3.5), s_o is (one of) the right-most ϵ-mode(s) of Σ^{cl}, and $w, z \in \mathbb{C}^{\tilde{n}}$ are non-zero vectors satisfying

$$w^*\phi(s_o) = 0 \ , \quad \phi(s_o)z = 0 \quad \text{and} \quad w^*z = 1,$$

where $\phi(s) := s\mathcal{E} - \bar{\mathcal{A}}(s)$ is the characteristic matrix of Σ^{cl}, where

$$\bar{\mathcal{A}}(s) := \mathcal{A}_0(p) + \sum_{i=1}^{\sigma} \mathcal{A}_i e^{-sh_i} \ , \tag{3.22}$$

and $\mathcal{E}, \mathcal{A}_i$, for $i = 0, \ldots, \sigma$, are as defined in (3.5).

Secondly, to find the partial derivatives of $\gamma_\psi(p^K)$, let $\mathcal{H}_0(p^K)$ and $\mathcal{H}_i$, $i = 1, \ldots, \sigma$, be the matrices in (3.13) and let

$$\Psi(p^K) := (\mathcal{H}_0(p^K))^{-1}\Psi_0 \tag{3.23}$$

where

$$\Psi_0 := \mathcal{H}_1 e^{-\psi h_1} \tag{3.24}$$

if $\sigma = 1$, and, if $\sigma \geq 2$,

$$\Psi_0 := \Psi_1(\theta^*) \tag{3.25}$$

where

$$\Psi_1(\theta) := \mathcal{H}_1 e^{-\psi h_1} + \sum_{k=2}^{\sigma} \mathcal{H}_k e^{-\psi h_k} e^{j\theta_k} \tag{3.26}$$

where $\theta := [\theta_2, \ldots, \theta_\sigma] \in [0, 2\pi]^{\sigma-1}$ and $\theta^* \in [0, 2\pi]^{\sigma-1}$ is such that

$$\rho\left(\left(\mathcal{H}_0(p^K)\right)^{-1} \Psi_1(\theta^*)\right) \geq \rho\left(\left(\mathcal{H}_0(p^K)\right)^{-1} \Psi_1(\theta)\right), \ \ \forall \theta \in [0, 2\pi]^{\sigma-1}.$$

Then, let γ^* be an eigenvalue of $\Psi(p^K)$ such that $|\gamma^*| = \rho(\Psi(p^K))$. Note that, $\gamma_\psi(p^K) = |\gamma^*|$. Furthermore, the partial derivatives of $\gamma_\psi(p^K)$ can be found as

$$\frac{\partial \gamma_\psi}{\partial p_k^K}(p^K) = \frac{1}{|\gamma^*|} \mathrm{Re}\left(\overline{\gamma^*} \hat{w}^* \left[\frac{\partial \Psi}{\partial p_k^K}(p^K)\right] \hat{z}\right) \tag{3.27}$$

where $\hat{w}, \hat{z} \in \mathbb{C}^{\hat{n}}$ are the vectors satisfying

$$\hat{w}^* \Psi(p^K) = \gamma^* \hat{w}^*, \quad \Psi(p^K)\hat{z} = \gamma^* \hat{z} \quad \text{and} \quad \hat{w}^* \hat{z} = 1.$$

Furthermore,

$$\frac{\partial \Psi}{\partial p_k^K}(p^K) = \frac{\partial (\mathcal{H}_0(p^K))^{-1}}{\partial p_k^K} \Psi_o \tag{3.28}$$

where

$$\frac{\partial (\mathcal{H}_0(p^K))^{-1}}{\partial p_k^K} = (\mathcal{H}_0(p^K))^{-1} \frac{\partial \mathcal{H}_0(p^K)}{\partial p_k^K} (\mathcal{H}_0(p^K))^{-1}. \tag{3.29}$$

3.4. Initialization of the Controller Parameters

The optimization algorithms of HANSO require an initial parameter vector, which corresponds to an initial controller in our case, as a starting point. In the software of [13], this vector is initialized randomly by a normal distribution centered at zero unless a specific initial parameter vector is indicated. However, since the

optimization problem to be solved is not convex, a completely random initialization may not always produce the desired result. Therefore, an initialization procedure which aims to facilitate the convergence of the optimization algorithm was developed in [22] and extended to the neutral case in [17]. Before outlining this procedure, let us first define the *blocking zeros* of the open-loop system Σ^c.

Definition 3.1. Let

$$T(s) := \bar{C}(s)\left(sE - \bar{A}(s)\right)^{-1}\bar{B}(s) + \bar{D}(s) \tag{3.30}$$

be the transfer function matrix (TFM) of Σ^c, where $\bar{A}(s) := \sum_{i=0}^{\sigma} A_i e^{-sh_i}$, $\bar{B}(s) := \sum_{i=0}^{\sigma} B_i e^{-sh_i}$, $\bar{C}(s) := \sum_{i=0}^{\sigma} C_i e^{-sh_i}$, and $\bar{D}(s) := \sum_{i=0}^{\sigma} D_i e^{-sh_i}$. Then, $\lambda \in \mathbb{C}$ is said to be a *blocking zero* of Σ^c, if $T(\lambda) = 0$.

It is well-known that, in order to stabilize a real LTI system by a real LTI controller, the controller must be chosen such that the open-loop system of the controller and the given system satisfies the so-called *parity interlacing property* (PIP) [47]. Therefore, in the initialization procedure of [22], the free parameters of the F matrix are chosen such that the open-loop system satisfies PIP.

In this procedure, first an $\epsilon < \mu$ is chosen. Then, a region $\mathcal{D}$ with the boundaries $\epsilon \leq \mathrm{Re}(\mathcal{D}) \leq \xi$ and $-\delta \leq \mathrm{Im}(\mathcal{D}) \leq \delta$, where ξ is the rightmost real ϵ-mode of Σ^c (if Σ^c does not have such a mode, then PIP is automatically satisfied and we take $l^{\min} = 0$) and δ is a small real number, is formed.

Then, the dynamics matrix of (3.3), F, is initialized such that it has one real eigenvalue between any pair of blocking zeros in the prescribed region $\mathcal{D}$ with an odd number of real modes of Σ^c between them and also one real eigenvalue to the right of the rightmost blocking zero in the region $\mathcal{D}$, if there is an odd number of real modes to the right of that zero. Here, the complex-conjugate pairs of blocking zeros within $\mathcal{D}$ are treated as real zeros in order not to force modes to move between such zeros (which would require high gains - see [22] for details). The dimension of the controller, l, is then initialized as $l^{\min}$, which is the number of such eigenvalues needed to satisfy PIP. If the dimension of the controller, l, has to be increased beyond $l^{\min}$ during the controller design algorithm, then $l - l^{\min}$ eigenvalues (either as real or as complex-conjugate pairs) are randomly chosen in the region $\mathbb{C}_\epsilon^-$. The

parameters of the F matrix are then determined such that F has those chosen eigenvalues (see Section 5.8 for details).

On the other hand, the free parameters of the G and H matrices are initialized randomly according to a normal distribution centered at zero. However, since Σ^{cl} may be a neutral system, $\gamma_\psi(p^K)$ may be greater than or equal to 1, which makes (3.18) infeasible from the very beginning. In order to avoid this situation, the elements of the K matrix are chosen randomly according to a normal distribution centered at zero so that $\gamma_\psi(p^K) < 1$. Otherwise, the magnitude of all the elements of K must be reduced until this condition is satisfied (by the assumption $C_D(\Sigma^c) < \psi$, this condition is satisfied for a small enough K) (see Section 5.8 for details).

3.5. Algorithm

In order to strongly μ-stabilize the given system Σ^c, for given μ and ψ, a controller of the form (3.3) can be designed by using the following algorithm.

Centralized Controller Design Algorithm:

1) Choose p_0^K randomly according to a normal distribution centered at zero so that $\gamma_\psi(p_0^K) < 1$.

2) Determine the minimum number of initial controller modes, indicated as $l^{\min}$, so that PIP is satisfied in the prescribed region $\mathcal{D}$. Let $l = l^{\min}$. If $l > 0$, choose the initial controller modes $s_1, \ldots, s_l$ so that the open-loop system satisfies PIP (see Section 3.4) and continue with the next step.

3) If $l > l^{\min}$, choose $s_{l^{\min}+1}, \ldots, s_l$ randomly in $\mathbb{C}_\epsilon^-$. Then, if $l > 0$, choose the free parameters of the F matrix such that F has eigenvalues $s_1, \ldots, s_l$. Then, choose the free parameters of G and H randomly according to a normal distribution centered at zero. Then, initialize the p vector accordingly, where the part of p that corresponds to the K matrix is p_0^K.

4) Solve the optimization problem (3.18). Let the minimizer be p^*.

5) If $c(p^*) < \mu$, **stop:** the controller corresponding to p^* strongly μ-stabilizes the given system Σ^c. Otherwise, let $l = l + 1$ and go back to step 3.

Since the optimization problem (3.18) is not convex, it is not guaranteed to end up with a p^* which corresponds to a strongly μ-stabilizing controller, i.e., it may happen that $c(p^*) \geq \mu$ no matter how large l is. However, let l^* be the minimum dimension so that there exists a controller of dimension l^* which stabilizes Σ^c (under the foregoing assumptions, there always exists such a l^* [30]). Then, empirical evidence has shown that, for some $l \geq l^*$, solving the optimization problem (3.18) for several starting points, which are chosen with a proper initialization procedure, leads to a desired result. Therefore, it is advised to repeat steps 3 and 4 by using different starting points (i.e., different $s_{l^{\min}+1}, \ldots, s_l$ values, different initial values for the G and H matrices, and different p_0^K, where each p_0^K satisfies the condition in step 1), before increasing the dimension of the controller in step 5.

4. DECENTRALIZED CONTROLLER DESIGN

Consider a decentralized LTI time-delay system Σ, with ν control agents, which is described by (2.24) where the autonomous part of Σ and (2.1) are in the same form. Furthermore, Σ describes a centralized system when $\nu = 1$, i.e., $\Sigma = \Sigma^c$. In order to μ-stabilize Σ, we consider finite-dimensional LTI output feedback decentralized controllers, $\mathcal{K}_1, \ldots, \mathcal{K}_\nu$, in the form of (2.30).

As it is stated in Section 2.3, μ-stabilizability of Σ is determined by its fixed modes. Given that $c_D(\Sigma) < \mu$, the system Σ can be μ-stabilized by the decentralized controllers (2.30) (with sufficiently large l_j, $j \in \bar{\nu}$) if and only if $\Lambda_\mu^d(\Sigma) = \emptyset$. Therefore, hereafter, it is assumed that Σ does not have any μ-DFMs. Furthermore, as it is emphasized in Section 2.1 and also taken into account in Chapter 3, the strong μ-stability, rather than μ-stability, is aimed. Therefore, it must be provided that $C_D(\Sigma) < \mu$.

The proposed method is based on the decentralized controller design algorithm of [22], where the controller design algorithm given in Section 3.5 is employed to design a centralized controller for each agent.

Suppose that, for some $s < \nu$, decentralized controllers $\mathcal{K}_1, \ldots, \mathcal{K}_s$ of the form (2.30) have been designed. Let Σ_s denote the decentralized control system with $\nu - s$ control agents which is obtained by applying the controllers $\mathcal{K}_1, \ldots, \mathcal{K}_s$ to the first s channels (in this notation Σ_0 denotes the open-loop system Σ). Let Σ_s^c denote the centralized system obtained from Σ_s by taking u_{s+1} as the only input and y_{s+1} as the only output. In order to describe Σ_s^c, let $m^s := \sum_{j=1}^s m_j$, $q^s := \sum_{j=1}^s q_j$ and $l^s := \sum_{j=1}^s l_j$. Also define $u^s(t) := [u_1^T(t) \ \cdots \ u_s^T(t)]^T \in \mathbb{R}^{m^s}$, $y^s(t) := [y_1^T(t) \ \cdots \ y_s^T(t)]^T \in \mathbb{R}^{q^s}$ and $z^s(t) := [z_1^T(t) \ \cdots \ z_s^T(t)]^T \in \mathbb{R}^{l^s}$. Then, the first s controllers can be compactly represented as

$$\begin{aligned} \dot{z}^s(t) &= F^s z^s(t) + G^s y^s(t) \\ u^s(t) &= H^s z^s(t) + K^s y^s(t) \end{aligned} \tag{4.1}$$

where

$$F^s := \text{bdiag}(F_1, ..., F_s) \ ,$$
$$G^s := \text{bdiag}(G_1, ..., G_s) \ ,$$
$$H^s := \text{bdiag}(H_1, ..., H_s) \ ,$$
$$K^s := \text{bdiag}(K_1, ..., K_s) \ .$$

Finally, define the "new state vector":

$$\eta_s(t) := \begin{bmatrix} x(t)^T & y^s(t)^T & z^s(t)^T & u^s(t)^T \end{bmatrix}^T \in \mathbb{R}^{n+m^s+q^s+l^s} \ . \tag{4.2}$$

Then, Σ_s^c can be described as

$$\begin{aligned} \hat{E}^s \dot{\eta}(t) &= \sum_{i=0}^{\sigma} \left(\hat{A}_i^s \eta_s(t-h_i) + \hat{B}_i^s u_{s+1}(t-h_i) \right) \\ y_{s+1}(t) &= \sum_{i=0}^{\sigma} \left(\hat{C}_i^s \eta_s(t-h_i) + D_{s+1,s+1,i} u_{s+1}(t-h_i) \right) \end{aligned} \tag{4.3}$$

where

$$\hat{E}^s := \begin{bmatrix} E & 0 & 0 & 0 \\ 0 & 0_{q^s} & 0 & 0 \\ 0 & 0 & I_{l^s} & 0 \\ 0 & 0 & 0 & 0_{m^s} \end{bmatrix}, \qquad \hat{A}_0^s := \begin{bmatrix} A_0 & 0 & 0 & B_0^s \\ C_0^s & -I_{q^s} & 0 & D_0^s \\ 0 & G^s & F^s & 0 \\ 0 & K^s & H^s & -I_{m^s} \end{bmatrix},$$

$$\hat{A}_i^s := \begin{bmatrix} A_i & 0 & 0 & B_i^s \\ C_i^s & 0_{q^s} & 0 & D_i^s \\ 0 & 0 & 0_{l^s} & 0 \\ 0 & 0 & 0 & 0_{m^s} \end{bmatrix}, \qquad i = 1, \ldots, \sigma$$

and, for $i = 0, \ldots, \sigma$,

$$\hat{B}_i^s := \begin{bmatrix} B_{s+1,i} \\ D_{1,s+1,i} \\ \vdots \\ D_{s,s+1,i} \\ 0_{(l^s+m^s)\times m^s} \end{bmatrix}, \qquad \hat{C}_i^s := \begin{bmatrix} C_{s+1,i} & 0_{q_{s+1}\times(q^s+l^s)} & D_{s+1,1,i} & \ldots & D_{s+1,s,i} \end{bmatrix}.$$

Note that, since (3.5) describes the autonomous part of (4.3) for $s = 1, \ldots, \nu$, the well-posed of Σ_s^c is guaranteed when K^s is chosen so that (3.11) is satisfied (or (3.12) when Σ_{s-1}^c is retarded).

The decentralized controller design algorithm designs a controller $\mathcal{K}_{s+1}$ of the form (2.30) to strongly μ-stabilize Σ_s^c, except for its μ-CFMs, for each $s = 1, \ldots, \nu - 1$, sequentially. Since each CFM of any Σ_s^c should appear as a non-fixed mode of Σ_r^c for some $r > s$ with probability one as long as it is not a DFM (follows from the arguments of [34] - see [22]), this strategy results in a strongly μ-stabilizing overall decentralized controller for Σ, as long as $C_D(\Sigma) < \mu$ and $\Lambda_\mu^d(\Sigma) = \emptyset$.

In the design of $\mathcal{K}_{s+1}$, for $s = 1, \ldots, \nu - 1$, the centralized controller design algorithm, which utilizes the nonsmooth optimization algorithms together with the initialization procedure given in Section 3.4, is employed. But, this algorithm must be modified when it is used in the decentralized controller design.

First of all, it is assumed that (3.1) is output feedback μ-stabilizable, i.e., besides $C_D(\Sigma^c) < \mu$, $\Lambda_\mu^c(\Sigma^c) = \emptyset$. However, in a decentralized framework, even though the overall system does not have any μ-DFMs, the system from a particular input channel to the corresponding output channel may have μ-CFMs. In the existence of a μ-CFM, the optimization procedure will be stuck at such a mode. Therefore, the optimization problem (3.18) must be modified so that only the modes, which are not CFMs of Σ^c are considered. From here on, we refer to these modes as the *non-fixed modes*. Thus, the spectral abscissa function, $c(\Sigma^{cl})$ is re-defined as

$$\bar{c}(\Sigma^{cl}) := \sup \left\{ \zeta \in \mathbb{R} \mid \Omega_\zeta(\Sigma^{cl}) \setminus \Lambda_\zeta^c(\Sigma_s^c) \neq \emptyset \right\} , \tag{4.4}$$

where Σ^{cl} denotes the closed-loop system obtained by applying the controller $\mathcal{K}_{s+1}$ to the system Σ_s^c.

Secondly, the initialization procedure given in Section 3.4 and the associated steps of the centralized controller design algorithm must be modified so that only the non-fixed modes are considered.

Therefore, $\mathcal{K}_{s+1}$ is designed such that $\bar{c}(\Sigma^{cl}) < \mu$ and $\gamma_\psi(\Sigma^{cl}) < 1$, for some $\psi \in \mathbb{R}$, satisfying $C_D(\Sigma) < \psi < \mu$. Note that, the requirement $\gamma_\psi(\Sigma^{cl}) < 1$ also guarantees the well-posedness of Σ^{cl}.

The overall decentralized controller design algorithm is given as follows.

Decentralized Controller Design Algorithm:

1) Fix the upper limits, $\hat{l}_1, \ldots, \hat{l}_\nu$ on the dimensions of the decentralized controllers.

2) Let $k = 0$. Let Σ_0^c indicate the centralized system obtained from Σ by taking u_1 as the only input and y_1 as the only output.

3) If Σ_k^c is strongly μ-stable, except for its μ-CFMs, choose a random non-zero $K_{k+1} \in \mathbb{R}^{m_{k+1} \times q_{k+1}}$ such that the closed-loop system obtained by applying the static output feedback $u_{k+1}(t) = K_{k+1} y_{k+1}(t)$ to Σ_k^c is strongly μ-stable, except for its μ-CFMs (by the continuity of the modes with respect to the feedback gains, there exists such a K_{k+1} - see [48]). Indicate this controller as $\mathcal{K}_{k+1}^*$ and go to step 7. Otherwise, continue with step 4.

4) Determine the minimum dimension, $l_{k+1}^{\min}$, of the controller for Σ_k^c so that the system obtained by cascading Σ_k^c by the controller satisfies PIP, except for the fixed-modes of Σ_k^c (see Section 3.4). Let $l_{k+1} = l_{k+1}^{\min}$. If $l_{k+1} > \hat{l}_{k+1}$, let $\hat{l}_{k+1} = l_{k+1} + l_{n\_step}$ where l_{n_step} is some predetermined positive integer.

5) Design l_{k+1}-dimensional controller, $\mathcal{K}$, which is structured as in Section 3.1, in order to make Σ_k^{cl} strongly μ-stable, except for its μ-CFMs. If such a controller is obtained indicate the corresponding controller $\mathcal{K}$ as $\mathcal{K}_{k+1}^*$ and go to step 7. Otherwise, continue with step 6.

6) If $l_{k+1} < \hat{l}_{k+1}$, let $l_{k+1} = l_{k+1} + 1$ and go to step 5. Otherwise, indicate the last obtained controller $\mathcal{K}$ as $\mathcal{K}_{k+1}^*$ and go to step 7.

7) Apply the controller $\mathcal{K}_{k+1}^*$ to the system Σ_k to obtain Σ_{k+1}. If $k = \nu - 1$, go to step 8. Otherwise, let Σ_{k+1}^c indicate the system obtained from Σ_{k+1} by taking u_{k+2} as the only input and y_{k+2} as the only output. Let $k = k + 1$ and go to step 3.

8) If the overall closed-loop system, Σ_ν^{cl} is strongly μ-stable, stop: the desired decentralized controller has been obtained as $\mathcal{K}_1^*, \ldots, \mathcal{K}_\nu^*$. Otherwise, go to step 9.

9) If Σ_1 has some non-fixed μ-modes, let $m = 1$. Otherwise, determine $m > 1$ such that Σ_1, ..., Σ_{m-1} are all μ-stable, except for their μ-CFMs, but Σ_m is not. Let $k = m - 1$, $\hat{l}_m = \hat{l}_m + l_{n\_step}$, and $l_m = l_m + 1$, and go to step 5.

In the first step of the above algorithm, upper limits are defined to avoid using unnecessarily high-dimensional controllers for the lower indexed control agents. The reason for applying a static output feedback controller in step 3, whenever Σ_k^c is μ-stable, except for its μ-CFMs, is to make sure that any μ-mode of Σ, which is not a μ-DFM, is a non-fixed mode of Σ_s^c, for some $s > k$ (so that it can be eventually moved towards $\mathbb{C}_\mu^-$). As indicated in [34], if such a feedback loop is not closed, some μ-modes may not appear as non-fixed modes of Σ_s^c for any s, even if they are not μ-DFMs.

5. SOFTWARE (DCD-TDS v1.0)

As a main objective, the software package DCD-TDS is developed to design centralized or decentralized controllers to stabilize time-delay systems. In particular, if the given time-delay system is centralized in the form of (3.1), then a centralized controller in the form of (3.3) is designed by applying the centralized controller design algorithm with all its utilities given in Chapter 3. Otherwise, if the given system is decentralized in the form of (2.24), then decentralized controllers in the form of (2.30) are designed by using the decentralized controller design algorithm which employs the centralized controller design algorithm. Furthermore, DCD-TDS has ability to analyze the spectral properties of a given time-delay system. It should be noted that, DCD-TDS makes use of the slightly customized version of related functions in the software package TDS_STABIL [13].

In the subsequent sections, the main modules of the DCD-TDS are introduced with the associated functions. However, other than these functions, there exist sub-functions which are used either in the analysis or design phase of the software. The complete collection of these functions can be found in the software package.

Since, it is not practical to manage all the functions of DCD-TDS, a user-friendly Graphical-User-Interface (GUI), which provides an efficient utilization, is developed. This tool makes use of the developed functions of DCD-TDS, systematically. In the last Section, the GUI is introduced.

5.1. System and Controller Definition

To define a system of the form (2.25), a MATLAB function, called *tds_create*, has been created. This function has the same name as the related function in the TDS_STABIL software [13], which was developed for centralized time-delay systems. In fact, the current function is a generalization of *tds_create* in TDS_STABIL to the decentralized case. In order to create the system object,

systematically, we consider the system in the form (2.25) as

$$\begin{aligned}\dot{\hat{x}}(t)+\sum_{i=1}^{m^e}\left(\hat{E}_i\dot{\hat{x}}(t-h_i^e)\right)&=\sum_{i=1}^{m^a}\left(\hat{A}_i\hat{x}(t-h_i^a)\right)+\sum_{j=1}^{\nu}\sum_{i=1}^{m_j^b}\left(\hat{B}_{(j,i)}u_j(t-h_{(j,i)}^b)\right)\\ y_j(t)&=\sum_{i=1}^{m_j^c}\left(\hat{C}_{(j,i)}\hat{x}(t-h_{(j,i)}^c)\right)+\sum_{k=1}^{\nu}\sum_{i=1}^{m_{(j,k)}^d}\left(\hat{D}_{(j,k,i)}u_k(t-h_{(j,k,i)}^d)\right),\quad j\in\bar{\nu}\end{aligned}\tag{5.1}$$

where t is the time variable, $\hat{x}(t)\in\mathbb{R}^{\tilde{n}}$ is the state vector at time t, and $u_j(t)\in\mathbb{R}^{m_j}$ and $y_j(t)\in\mathbb{R}^{q_j}$ are, respectively, the input and the output vectors at time t, accessible by the j^{th} control agent ($j\in\bar{\nu}$). The matrices $\hat{E}_i$, $\hat{A}_i$, $\hat{B}_{(j,i)}$, $\hat{C}_{(j,i)}$ and $\hat{D}_{(j,k,i)}$ are constant real matrices. The number of distinct time-delays involved in the derivative of the state, in the state, in the inputs, at the outputs, and in the direct connections from the inputs to the outputs are indicated by the non-negative integers m^e, m^a, m_j^b, m_j^c and $m_{(j,k)}^d$ ($j\in\bar{\nu}$, $k\in\bar{\nu}$), respectively. Finally, $h_i^e\in(0,h^{\max}]$, $h_i^a,h_{(j,i)}^b,h_{(j,i)}^c,h_{(j,k,i)}^d\in[0,h^{\max}]$ are the time-delays, where $h^{\max}\geq 0$ is the maximum time-delay in the system. Note that h_i^e is positive for all $i\in\overline{m^e}$. However, some of h_i^a, $h_{(j,i)}^b$, $h_{(j,i)}^c$, and $h_{(j,k,i)}^d$ may be zero for some i,j,k, which would correspond to the delay-free part.

According to system definition (5.1), the function can be called in two different ways. For a retarded decentralized control system (i.e., when $m^e=0$), it is called as

$$\begin{aligned}\gg\ & tds = tds\_create(A, hA, B1, hB1, ..., Bn, hBn, C1, hC1, ..., Cn, hCn,\\ & D11, hD11, ..., D1n, hD1n, ..., Dn1, hDn1, ..., Dnn, hDnn, metadata)\end{aligned}$$

For a neutral decentralized control system (i.e., when $m^e\neq 0$), on the other hand, it is called as

$$\begin{aligned}\gg\ & tds = tds\_create(E, hE, A, hA, B1, hB1, ..., Bn, hBn, C1, hC1, ..., Cn, hCn,\\ & D11, hD11, ..., D1n, hD1n, ..., Dn1, hDn1, ..., Dnn, hDnn, metadata)\end{aligned}$$

Here, *metadata* is a struct which consist of several fields. These fields define certain properties of the system object. In particular, before calling the above function, two properties of the system must be predefined as

$$\gg metadata.Nagent = \nu;$$

$$\gg metadata.systype = 'id';$$

where the first field of *metadata* specifies the number of control agents (i.e., ν) and the second one is used as an identifier of the system type ($'retarded'$ for a retarded system and $'neutral'$ for a neutral system). Furthermore, the cell arrays of system matrices and vectors of system delays are defined as follows:

$$
\begin{aligned}
E &= \{\hat{E}_1,\ \dots\ , \hat{E}_{m^e}\},\\
hE &= [h^e_1,\ \dots\ , h^e_{m^e}],\\
A &= \{\hat{A}_1,\ \dots\ , \hat{A}_{m^a}\},\\
hA &= [h^a_1,\ \dots\ , h^a_{m^a}],\\
B1 &= \{\hat{B}_{(1,1)},\ \dots\ , \hat{B}_{(1,m^b_1)}\},\\
hB1 &= [h^b_{(1,1)},\ \dots\ , h^b_{(1,m^b_1)}],\\
&\vdots\\
Bn &= \{\hat{B}_{(\nu,1)},\ \dots\ , \hat{B}_{(\nu,m^b_\nu)}\},\\
hBn &= [h^b_{(\nu,1)},\ \dots\ , h^b_{(\nu,m^b_\nu)}],\\
C1 &= \{\hat{C}_{(1,1)},\ \dots\ , \hat{C}_{(1,m^c_1)}\},\\
hC1 &= [h^c_{(1,1)},\ \dots\ , h^c_{(1,m^c_1)}],\\
&\vdots\\
Cn &= \{\hat{C}_{(\nu,1)},\ \dots\ , \hat{C}_{(\nu,m^c_\nu)}\},\\
hCn &= [h^c_{(\nu,1)},\ \dots\ , h^c_{(\nu,m^c_\nu)}],\\
D11 &= \{\hat{D}_{(1,1,1)},\ \dots\ , \hat{D}_{(1,1,m^d_{(1,1)})}\},\\
hD11 &= [h^d_{(1,1,1)},\ \dots\ , h^d_{(1,1,m^d_{(1,1)})}],\\
\vdots D1n &= \{\hat{D}_{(1,\nu,1)},\ \dots\ , \hat{D}_{(1,\nu,m^d_{(1,\nu)})}\},\\
hD1n &= [h^d_{(1,\nu,1)},\ \dots\ , h^d_{(1,\nu,m^d_{(1,\nu)})}],\\
&\vdots\\
Dn1 &= \{\hat{D}_{(\nu,1,1)},\ \dots\ , \hat{D}_{(\nu,1,m^d_{(\nu,1)})}\},\\
hDn1 &= [h^d_{(\nu,1,1)},\ \dots\ , h^d_{(\nu,1,m^d_{(\nu,1)})}],\\
&\vdots
\end{aligned}
$$

$$Dnn = \{\hat{D}_{(\nu,\nu,1)},\ \dots\ , \hat{D}_{(\nu,\nu,m^d_{(\nu,\nu)})}\},$$
$$hDnn = [h^d_{(\nu,\nu,1)},\ \dots\ , h^d_{(\nu,\nu,m^d_{(\nu,\nu)})}]\ .$$

It should be noted that, the time-delays in (5.1) with same superscript must be distinct. Therefore, even if the user enter same delays with same superscript while defining (5.1), this function sums up all the matrices which corresponds to the same delay, hence, creates a system object in the form (2.25) with distinct delays.

Following the same procedure given in Section 2.3, by defining $\delta_e(t) := \hat{x}(t) + \sum_{i=1}^{m^e} \left(\hat{E}_i \hat{x}(t - h^e_i)\right)$ and $\underline{\mathrm{x}}(t) := \begin{bmatrix} \delta_e(t)^T & \hat{x}(t)^T \end{bmatrix}^T \in \mathbb{R}^n$, the system, described by (5.1), can alternatively be rewritten as

$$\begin{aligned} \underline{\mathrm{E}}\dot{x}(t) &= \underline{\mathrm{A}}_0 \underline{\mathrm{x}}(t) + \sum_{i=1}^{m^{ae}} \left(\underline{\mathrm{A}}_i \underline{\mathrm{x}}(t - h^{ae}_i)\right) + \sum_{j=1}^{\nu} \sum_{i=1}^{m^b_j} \left(\underline{\mathrm{B}}_{(j,i)} u_j(t - h^b_{(j,i)})\right) \\ y_j(t) &= \sum_{i=1}^{m^c_j} \left(\underline{\mathrm{C}}_{(j,i)} \underline{\mathrm{x}}(t - h^c_{(j,i)})\right) + \sum_{k=1}^{\nu} \sum_{i=1}^{m^d_{(j,k)}} \left(\underline{\mathrm{D}}_{(j,k,i)} u_k(t - h^d_{(j,k,i)})\right)\ ,\quad j \in \overline{\nu} \end{aligned} \tag{5.2}$$

where

$$\{h^{ae}_1, \dots, h^{ae}_{m^{ae}}\} := \{\{h^a_1, \dots, h^a_{m^a}\} \backslash \{0\}\} \cup \{h^e_1, \dots, h^e_{m^e}\}\ ,$$

$$\underline{\mathrm{E}} := \begin{bmatrix} I_{\tilde{n}} & 0 \\ 0 & 0_{\tilde{n}\times\tilde{n}} \end{bmatrix}\ ,\quad \underline{\mathrm{A}}_0 := \begin{bmatrix} 0 & \hat{\hat{A}}_0 \\ -I_{\tilde{n}} & I_{\tilde{n}} \end{bmatrix}\ ,\quad \underline{\mathrm{A}}_i := \begin{bmatrix} 0 & \hat{\hat{A}}_i \\ 0 & \hat{\hat{E}}_i \end{bmatrix}\ ,\ i = 1, \dots, m^{ae}\ ,$$

and, for $j, k \in \bar{\nu}$,

$$\begin{aligned} \underline{\mathrm{B}}_{(j,i)} &:= \begin{bmatrix} \hat{B}_{(j,i)} \\ 0_{\hat{n}\times p_j} \end{bmatrix}\ ,\quad i = 1, \dots, m^{m^b_j}\ , \\ \underline{\mathrm{C}}_{(j,i)} &:= \begin{bmatrix} 0_{q_j\times\hat{n}} & \hat{C}_{(j,i)} \end{bmatrix}\ ,\quad i = 1, \dots, m^{m^c_j}\ , \\ \underline{\mathrm{D}}_{(j,k,i)} &:= \hat{D}_{(j,k,i)}\ ,\quad i = 1, \dots, m^{m^d_j} \end{aligned}$$

where

$$\hat{\hat{A}}_0 := \begin{cases} \hat{A}_r, & \text{if } \mathrm{h^a_r} = 0 \\ 0_{\hat{n}\times\hat{n}}, & \text{if } 0 \notin \{\mathrm{h^a_1}, \dots, \mathrm{h^a_{m^a}}\} \end{cases}\ ,$$

$$\hat{\hat{A}}_i := \begin{cases} \hat{A}_r, & \text{if } \mathrm{h}_{\mathrm{i}}^{\mathrm{ae}} = \mathrm{h}_{\mathrm{r}}^{\mathrm{a}} \\ 0_{\hat{n}\times\hat{n}}, & \text{if } \mathrm{h}_{\mathrm{i}}^{\mathrm{ae}} \notin \{\mathrm{h}_1^{\mathrm{a}}, \ldots, \mathrm{h}_{\mathrm{m^a}}^{\mathrm{a}}\} \end{cases}, \quad i = 1, \ldots, m^{ae} ,$$

and

$$\hat{\hat{E}}_i := \begin{cases} \hat{E}_r, & \text{if } \mathrm{h}_{\mathrm{i}}^{\mathrm{ae}} = \mathrm{h}_{\mathrm{r}}^{\mathrm{e}} \\ 0_{\hat{n}\times\hat{n}}, & \text{if } \mathrm{h}_{\mathrm{i}}^{\mathrm{ae}} \notin \{\mathrm{h}_1^{\mathrm{e}}, \ldots, \mathrm{h}_{\mathrm{m^e}}^{\mathrm{e}}\} \end{cases}, \quad i = 1, \ldots, m^{ae} .$$

As it is stated in Section 2.3, the form (5.2) is more convenient to work with. Therefore, in the software, the system in the form (5.1) is first converted to the form (5.2). The function developed for this purpose is called as

$$\gg \quad openDDAE = tds2DDAE(tds);$$

where *tds* is the system object in the form (5.1) and *openDDAE* is the one in the form (5.2).

Furthermore, a controller object, which corresponds to (3.3), may need to be created as well. The function which is developed to create controller object is named as *controller_create*. To define a static controller, it can be called as

$$\gg \quad controller = controller\_create(K)$$

where *controller* is the created controller object and K is the control gain matrix. To define a dynamic controller, it can be called as

$$\gg \quad controller = controller\_create(F, G, H, K)$$

where F, G, H, and K are, respectively, the dynamics, the input, the output, and the static gain matrices of the controller.

5.2. Closed-Loop System Definition

In this module, the functions which are used to obtain closed-loop system objects are presented. Now, suppose that s ($s \leq \nu$) controllers, $controllller_1$, ..., $controller_s$, of the form (2.30), have been obtained for a decentralized control system with ν control agents. The decentralized controller design algorithm requires obtaining a description of the decentralized control system, to be denoted by Σ_s, with the remaining $\nu - s$ control agents after the first s loops have been closed.

The associated system object of Σ_s, obtained by closing the first s loops with $controllller_1$, ..., $controller_s$, which can be compactly represented by (4.1), can be described, by defining a new state vector

$$\eta_s(t) := \begin{bmatrix} \underline{\mathrm{x}}(t)^T & y^s(t)^T & z^s(t)^T & u^s(t)^T \end{bmatrix}^T , \tag{5.3}$$

as follows:

$$\begin{aligned} &\begin{bmatrix} \underline{\mathrm{E}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & I_{l^s} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \dot{\eta}_s(t) = \begin{bmatrix} \underline{\mathrm{A}}_0 & 0 & 0 & \hat{\underline{\mathrm{B}}}_0^s \\ \hat{\underline{\mathrm{C}}}_0^s & -I_{q^s} & 0 & \hat{\underline{\mathrm{D}}}_0^s \\ 0 & G^s & F^s & 0 \\ 0 & K^s & H^s & -I_{p^s} \end{bmatrix} \eta_s(t) \\ &+ \sum_{i=1}^{m^s} \begin{bmatrix} \hat{\underline{\mathrm{A}}}_i & 0 & 0 & \hat{\underline{\mathrm{B}}}_i^s \\ \hat{\underline{\mathrm{C}}}_i^s & 0 & 0 & \hat{\underline{\mathrm{D}}}_i^s \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \eta_s(t - h_i^s) + \sum_{j=s+1}^{\nu} \sum_{i=1}^{\hat{m}_j^b} \hat{\underline{\mathrm{B}}}_{(j,i)} u_j(t - \hat{h}_{(j,i)}^b) \\ &y_j(t) = \sum_{i=1}^{\hat{m}_j^c} \hat{\underline{\mathrm{C}}}_{(j,i)} \eta_s(t - \hat{h}_i^c) + \sum_{k=s+1}^{\nu} \sum_{i=1}^{m_{(j,k)}^d} \underline{\mathrm{D}}_{(j,k,i)} u_k(t - h_{(j,k,i)}^d), \end{aligned} \tag{5.4}$$

$j = s+1, \ldots, \nu$, where

$$\hat{\underline{\mathrm{B}}}_0^s := \begin{bmatrix} \hat{\hat{\underline{\mathrm{B}}}}_{(1,0)} & \cdots & \hat{\hat{\underline{\mathrm{B}}}}_{(s,0)} \end{bmatrix} , \quad \hat{\underline{\mathrm{C}}}_0^s := \begin{bmatrix} \hat{\hat{\underline{\mathrm{C}}}}_{(1,0)}^T & \cdots & \hat{\hat{\underline{\mathrm{C}}}}_{(s,0)}^T \end{bmatrix}^T ,$$

$$\hat{\underline{\mathrm{D}}}_0^s := \begin{bmatrix} \hat{\hat{\underline{\mathrm{D}}}}_{1,1,0}, & \cdots & \hat{\hat{\underline{\mathrm{D}}}}_{1,s,0} \\ \vdots & \ddots & \vdots \\ \hat{\hat{\underline{\mathrm{D}}}}_{s,1,0} & \cdots & \hat{\hat{\underline{\mathrm{D}}}}_{s,s,0} \end{bmatrix} ,$$

where, for $j, k \in \bar{\nu}$,

$$\hat{\hat{\underline{\mathrm{B}}}}_{(j,0)} := \begin{cases} \underline{\mathrm{B}}_{(j,r)}, & \text{if } \mathrm{h}_{(\mathrm{j,r})}^{\mathrm{b}} = 0 \\ 0_{n \times p_j}, & \text{if } 0 \notin \{\mathrm{h}_{(\mathrm{j},1)}^{\mathrm{b}}, \ldots, \mathrm{h}_{(\mathrm{j},\mathrm{m}_{\mathrm{j}}^{\mathrm{b}})}^{\mathrm{b}}\} \end{cases} ,$$

$$\hat{\hat{\underline{\mathrm{C}}}}_{(j,0)} := \begin{cases} \underline{\mathrm{C}}_{(j,r)}, & \text{if } \mathrm{h}_{(\mathrm{j,r})}^{\mathrm{c}} = 0 \\ 0_{q_j \times n}, & \text{if } 0 \notin \{\mathrm{h}_{(\mathrm{j},1)}^{\mathrm{c}}, \ldots, \mathrm{h}_{(\mathrm{j},\mathrm{m}_{\mathrm{j}}^{\mathrm{c}})}^{\mathrm{c}}\} \end{cases} ,$$

$$\hat{\hat{\underline{\mathrm{D}}}}_{(j,k,0)} := \begin{cases} \underline{\mathrm{D}}_{(j,k,r)}, & \text{if } \mathrm{h}_{(\mathrm{j,k,r})}^{\mathrm{d}} = 0 \\ 0_{q_j \times p_j}, & \text{if } 0 \notin \{\mathrm{h}_{(\mathrm{j,k},1)}^{\mathrm{d}}, \ldots, \mathrm{h}_{(\mathrm{j,k},\mathrm{m}_{(\mathrm{j,k})}^{\mathrm{d}})}^{\mathrm{d}}\} \end{cases} ,$$

and, for $i = 1, \ldots, m^s$,

$$\hat{\underline{\mathrm{A}}}_i := \begin{cases} \underline{\mathrm{A}}_r, & \text{if } \mathrm{h}^{\mathrm{s}}_{\mathrm{i}} = \mathrm{h}^{\mathrm{ae}}_{\mathrm{r}} \\ 0_{n\times n}, & \text{if } \mathrm{h}^{\mathrm{s}}_{\mathrm{i}} \notin \{\mathrm{h}^{\mathrm{ae}}_1, \ldots, \mathrm{h}^{\mathrm{ae}}_{\mathrm{m^{ae}}}\} \end{cases},$$

$$\hat{\underline{\mathrm{B}}}^s_i := \begin{bmatrix} \hat{\underline{\mathrm{B}}}_{(1,i)} & \cdots & \hat{\underline{\mathrm{B}}}_{(s,i)} \end{bmatrix}, \quad \hat{\underline{\mathrm{C}}}^s_i := \begin{bmatrix} \hat{\underline{\mathrm{C}}}^T_{(1,i)} & \cdots & \hat{\underline{\mathrm{C}}}^T_{(s,i)} \end{bmatrix}^T,$$

and

$$\hat{\underline{\mathrm{D}}}^s_i := \begin{bmatrix} \hat{\underline{\mathrm{D}}}_{(1,1,i)}, & \cdots & \hat{\underline{\mathrm{D}}}_{(1,s,i)} \\ \vdots & \ddots & \vdots \\ \hat{\underline{\mathrm{D}}}_{(s,1,i)} & \cdots & \hat{\underline{\mathrm{D}}}_{(s,s,i)} \end{bmatrix},$$

where, for $j, k \in \overline{s}$,

$$\hat{\underline{\mathrm{B}}}_{(j,i)} := \begin{cases} \underline{\mathrm{B}}_{(j,r)}, & \text{if } \mathrm{h}^{\mathrm{s}}_{\mathrm{i}} = \mathrm{h}^{\mathrm{b}}_{(\mathrm{j,r})} \\ 0_{n\times p_j}, & \text{if } \mathrm{h}^{\mathrm{s}}_{\mathrm{i}} \notin \{\mathrm{h}^{\mathrm{b}}_{(\mathrm{j},1)}, \ldots, \mathrm{h}^{\mathrm{b}}_{(\mathrm{j},\mathrm{m}^{\mathrm{b}}_{\mathrm{j}})}\} \end{cases},$$

$$\hat{\underline{\mathrm{C}}}_{(j,i)} := \begin{cases} \underline{\mathrm{C}}_{(j,r)}, & \text{if } \mathrm{h}^{\mathrm{s}}_{\mathrm{i}} = \mathrm{h}^{\mathrm{c}}_{(\mathrm{j,r})} \\ 0_{q_j\times n}, & \text{if } \mathrm{h}^{\mathrm{s}}_{\mathrm{i}} \notin \{\mathrm{h}^{\mathrm{c}}_{(\mathrm{j},1)}, \ldots, \mathrm{h}^{\mathrm{c}}_{(\mathrm{j},\mathrm{m}^{\mathrm{c}}_{\mathrm{j}})}\} \end{cases},$$

and

$$\hat{\underline{\mathrm{D}}}_{(j,k,i)} := \begin{cases} \underline{\mathrm{D}}_{(j,k,r)}, & \text{if } \mathrm{h}^{\mathrm{s}}_{\mathrm{i}} = \mathrm{h}^{\mathrm{d}}_{(\mathrm{j,k,r})} \\ 0_{q_j\times p_j}, & \text{if } \mathrm{h}^{\mathrm{s}}_{\mathrm{i}} \notin \{\mathrm{h}^{\mathrm{d}}_{(\mathrm{j,k},1)}, \ldots, \mathrm{h}^{\mathrm{d}}_{(\mathrm{j,k},\mathrm{m}^{\mathrm{d}}_{(\mathrm{j,k})})}\} \end{cases}.$$

Furthermore, for $j = s+1, \ldots, \nu$,

$$\hat{\underline{\mathrm{B}}}_{(j,i)} := \begin{bmatrix} \hat{\underline{\mathrm{B}}}^T_{(j,i)} & \hat{\underline{\mathrm{D}}}^T_{(1,j,i)} & \cdots & \hat{\underline{\mathrm{D}}}^T_{(s,j,i)} & 0_{p_j\times(l^s+p^s)} \end{bmatrix}^T$$

and

$$\hat{\underline{\mathrm{C}}}_{(j,i)} := \begin{bmatrix} \hat{\underline{\mathrm{C}}}_{(j,i)} & 0_{q_j\times(q^s+l^s)} & \hat{\underline{\mathrm{D}}}_{(j,1,i)} & \cdots & \hat{\underline{\mathrm{D}}}_{(j,s,i)} \end{bmatrix},$$

where, for $i = 1, \ldots, \hat{m}^b_j$,

$$\hat{\hat{\underline{\mathrm{B}}}}_{(j,i)} := \begin{cases} \underline{\mathrm{B}}_{(j,r)}, & \text{if } \hat{\mathrm{h}}^{\mathrm{b}}_{(\mathrm{j,i})} = \mathrm{h}^{\mathrm{b}}_{(\mathrm{j,r})} \\ 0_{n\times p_j}, & \text{if } \hat{\mathrm{h}}^{\mathrm{b}}_{(\mathrm{j,i})} \notin \{\mathrm{h}^{\mathrm{b}}_{(\mathrm{j},1)}, \ldots, \mathrm{h}^{\mathrm{b}}_{(\mathrm{j},\mathrm{m}^{\mathrm{b}}_{\mathrm{j}})}\} \end{cases},$$

and, for $k \in \overline{s}$,

$$\hat{\underline{\mathrm{D}}}_{(k,j,i)} := \begin{cases} \underline{\mathrm{D}}_{(k,j,r)}, & \text{if } \hat{\mathrm{h}}^{\mathrm{b}}_{(\mathrm{j,i})} = \mathrm{h}^{\mathrm{d}}_{(\mathrm{k,j,r})} \\ 0_{q_k \times p_j}, & \text{if } \hat{\mathrm{h}}^{\mathrm{b}}_{(\mathrm{j,i})} \notin \{\mathrm{h}^{\mathrm{d}}_{(\mathrm{k,j,1})}, \ldots, \mathrm{h}^{\mathrm{d}}_{(\mathrm{k,j,m}^{\mathrm{d}}_{(\mathrm{k,j})})}\} \end{cases},$$

and, for $i = 1, \ldots, \hat{m}^c_j$,

$$\hat{\underline{\mathrm{C}}}_{(j,i)} := \begin{cases} \underline{\mathrm{C}}_{(j,r)}, & \text{if } \hat{\mathrm{h}}^{\mathrm{c}}_{(\mathrm{j,i})} = \mathrm{h}^{\mathrm{c}}_{(\mathrm{j,r})} \\ 0_{q_j \times n}, & \text{if } \hat{\mathrm{h}}^{\mathrm{c}}_{(\mathrm{j,i})} \notin \{\mathrm{h}^{\mathrm{c}}_{(\mathrm{j,1})}, \ldots, \mathrm{h}^{\mathrm{c}}_{(\mathrm{j,m}^{\mathrm{c}}_{\mathrm{j}})}\} \end{cases},$$

and, for $k \in \overline{s}$,

$$\hat{\underline{\mathrm{D}}}_{(j,k,i)} := \begin{cases} \underline{\mathrm{D}}_{(j,k,r)}, & \text{if } \hat{\mathrm{h}}^{\mathrm{c}}_{(\mathrm{j,i})} = \mathrm{h}^{\mathrm{d}}_{(\mathrm{j,k,r})} \\ 0_{q_j \times p_k}, & \text{if } \hat{\mathrm{h}}^{\mathrm{c}}_{(\mathrm{j,i})} \notin \{\mathrm{h}^{\mathrm{d}}_{(\mathrm{j,k,1})}, \ldots, \mathrm{h}^{\mathrm{d}}_{(\mathrm{j,k,m}^{\mathrm{d}}_{(\mathrm{j,k})})}\} \end{cases}.$$

Also, it is defined that

$$\{h^s_1, \ldots, h^s_{m^s}\} := \left\{\{h^{ae}_1, \ldots, h^{ae}_{m^{ae}}\} \cup \bar{h}^b_s \cup \bar{h}^c_s \cup \bar{h}^d_s\right\} \setminus \{0\}$$

where

$$\bar{h}^b_s := \bigcup_{j=1}^{s} \left\{h^b_{(j,1)}, \ldots, h^b_{(j,m^b_j)}\right\},$$

$$\bar{h}^c_s := \bigcup_{j=1}^{s} \left\{h^c_{(j,1)}, \ldots, h^c_{(j,m^c_j)}\right\},$$

and

$$\bar{h}^d_s := \bigcup_{j=1}^{s} \bigcup_{k=1}^{s} \left\{h^b_{(j,k,1)}, \ldots, h^d_{\left(j,k,m^d_{(j,k)}\right)}\right\}.$$

Furthermore, for $j = s+1, \ldots, \nu$, define

$$\left\{\hat{h}^b_{(j,1)}, \ldots, \hat{h}^b_{(j,\hat{m}^b_j)}\right\} := \left\{h^b_{(j,1)}, \ldots, h^b_{(j,m^b_j)}\right\} \bigcup \left\{\bigcup_{k=1}^{s} \left\{h^d_{(k,j,1)}, \ldots, h^d_{(k,j,m^d_{(k,j)})}\right\}\right\}$$

and

$$\left\{\hat{h}^c_{(j,1)}, \ldots, \hat{h}^c_{(j,\hat{m}^c_j)}\right\} := \left\{h^c_{(j,1)}, \ldots, h^c_{(j,m^c_j)}\right\} \bigcup \left\{\bigcup_{k=1}^{s} \left\{h^d_{(j,k,1)}, \ldots, h^d_{(j,k,m^d_{(j,k)})}\right\}\right\}.$$

The function *loop_close* creates the system described by (5.4). It can be called as

$$\gg \quad res\_DDAE = loop\_close(openDDAE, controller_1, \ldots, controller_s)$$

Here, *res_DDAE* is the resulting system and *openDDAE* is the original system. By this way, for $s = \nu$, the overall closed-loop system can ben obtained. Thus, the overall closed-loop system is described as

$$\begin{bmatrix} \underline{\mathrm{E}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & I_{l^\nu} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \dot{\eta}_\nu(t) = \begin{bmatrix} \hat{\underline{\mathrm{A}}}_0 & 0 & 0 & \hat{\underline{\mathrm{B}}}_0^\nu \\ \hat{\underline{\mathrm{C}}}_0^\nu & -I_{q^\nu} & 0 & \hat{\underline{\mathrm{D}}}_0^\nu \\ 0 & \hat{G}^\nu & \hat{F}^\nu & 0 \\ 0 & \hat{K}^\nu & \hat{H}^\nu & -I_{p^\nu} \end{bmatrix} \eta_\nu(t) + \sum_{i=1}^{m^\nu} \begin{bmatrix} \hat{\underline{\mathrm{A}}}_i & 0 & 0 & \hat{\underline{\mathrm{B}}}_i^\nu \\ \hat{\underline{\mathrm{C}}}_i^\nu & 0 & 0 & \hat{\underline{\mathrm{D}}}_i^\nu \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \eta_\nu(t - h_i^\nu) \tag{5.5}$$

We note that in the case $l_j = 0$, for some or all $j \in \bar{\nu}$, the descriptions given in (5.4) and (5.5) are still valid, except that in this case $z_j(t)$ would be missing in (5.3) and the matrices F_j, G_j, and H_j would be missing in (5.4) and (5.5).

Furthermore, considering a centralized system (5.2) (for $\nu = 1$) and a controller in the form of (3.3), where all the entries of the controller matrices is zero, one can obtain the closed-loop system by calling

$$\gg \quad closedDDAE = getclosedDDAE(openDDAE, Nz)$$

where *openDDAE* is the system object created by the function *tds2DDAE*, Nz is the dimension of the controller (i.e., l), and *closedDDAE* is the closed-loop system object in the form of (3.5). Once such a closed-loop system, which can be considered as a template since the entries of the controller matrices are all zero, is obtained, one can obtain the true closed-loop system by placing the free parameters of the controller in their exact places according to the controller structure used (see Section 5.7).

5.3. Computation of ϵ-modes

This module includes functions to compute the set of the ϵ-modes, defined as in Section 2.1, of a system whose autonomous part can be described by (2.1).

Such modes, however, can be computed only when there are finitely many of them. As explained in Section 2.1, such would be the case if $\epsilon > c_D$, where c_D is given by (2.10). However, as discussed in Section 2.1, c_D is not robust to perturbations in the delays. Therefore, we use C_D instead of c_D (see Section 5.6). Once C_D is computed, the ϵ-modes, for any $\epsilon > C_D$, of (2.1) can be computed by the *spectral discretization approach* of [13], which extends the approach of [49] to systems whose autonomous part can be described by DDAEs.

The present function used for this purpose is only a slightly customized version of the function of TDS_STABIL software. This function is named as *compute_roots_DDAE* and can be called as

$$\gg \quad eigenvalues = compute\_roots\_DDAE(ddae, rootoptions)$$

where *ddae* is the system object whose autonomous part is in the form of (2.1) (e.g., created by the *tds_create* function and transformed to the form (5.2) by using $tds2DDAE$ function - see Section 5.1) and *eigenvalues* is the set of modes computed. The *rootoptions* are same as in the TDS_STABIL software and are defined as follows:

- **minimal_real_part** is the ϵ value for which ϵ-modes are to be computed. The default value is -1.
- **max_size_eigenvalue_problem** is the limit of the number of discretization points for the characteristic roots computations. The default value is 1000.
- **newton_max_iterations** is the maximum number of Newton iterations to correct characteristic roots. The default value is 20.
- **root_accuracy** is the norm on the residual which is used as the stopping criterion for the Newton's method. The default value is 10^{-10}.
- **commensurate_basic_delay** is the basic delay h in the case of commensurate delays. The use of this option is optional and, if used, may speed up the computations if all the delays are commensurate.
- **number_grid_points_p** is the number of grid points in the $[0, 2\pi)$ interval for the discretization. The default value is 20.

- **new_basic_delay_q** is used for approximating delays by commensurate delays. The default value is 100.

Although there are infinitely many ϵ-modes, for $\epsilon < C_D$, there will be finitely many ϵ-modes for any finite $\bar{\epsilon}$ which satisfies $-\bar{\epsilon} \leq Im(s) \leq \bar{\epsilon}$. Relying on that, the ϵ-modes of (2.1) can be computed in a predetermined region. For this purpose, *quasi-polynomial mapping based root-finder* (QPmR) algorithm of [50] is used. This algorithm is developed for computing all the roots of a quasi-polynomial located in a predefined bounded region $\mathcal{R}$ of the complex plane. The input arguments of QPMR are the function whose roots are to be found and the region $\mathcal{R}$ specified by its minimum and maximum boundaries of its real and imaginary parts. The approach behind this function is based on mapping the real and imaginary parts of the quasi-polynomial in the complex plane using the level curve tracing algorithm [51]. In order to compute the ϵ-modes of (2.1) in a predetermined region, a function named as $QPmR\_roots$ can be called as

$$\gg \quad eigenvalues = QPmR\_roots(ddae, options)$$

where *ddae* is the system object whose autonomous part is in the form of (2.1) and *eigenvalues* is the set of modes computed. The *qpmroptions* are the options of QPmR itself and are defined as follows:

- **region** is a vector, whose elements specify the boundaries of the region, and in the form of $[real\_min\ real\_max\ imag\_min\ imag\_max]$, where the first two elements indicate the left and right hand side boundaries, respectively, and the last two elements indicate the upper and lower boundaries, respectively.
- **e** specifies the computation accuracy. If e=-1, then e=10^{-3}*ds.
- **ds** specifies the grid step for mapping the zero-level curves. If ds=-1, the grid step is adjusted automatically.

5.4. Computation of ϵ-fixed modes

As it is indicated in Section 2.3, given that $\mu > c_D$, in order to determine whether or not it is possible to μ-stabilize a system of the form (2.24), it suffices to compute $\Lambda_\epsilon^d(\Sigma)$, for some $\epsilon < \mu$.

There are two main methods to compute $\Lambda_\epsilon^d(\Sigma)$. First one is called as *rank test*, which is based on the rank test given by (2.32) in Lemma 1. In order to determine $\Lambda_\epsilon^d(\Sigma)$, $\Omega_\epsilon(\Sigma)$ must first be obtained. Then, for each $s_0 \in \Omega_\epsilon(\Sigma)$, the rank test in Lemma 1 must be carried out to determine whether $s_0 \in \Lambda_\epsilon^d(\Sigma)$.

The other method to compute $\Lambda_\epsilon^d(\Sigma)$ is named as *numerical procedure* and is based on the algorithm, which was originally presented in [34] for finite-dimensional systems. In this method, different static output feedback controllers are generated by using a random number generator, then, the ϵ-modes of the closed-loop systems are computed in order to determine whether a certain ϵ-mode remains fixed or not. Here, the closed-loop system is obtained by using the functions given in Section 5.2.

It should be noted that, the numerical procedure gives $\Lambda_\epsilon^d(\Sigma)$ with probability 1. Therefore, the rank test is more certain than the numerical procedure as long as the rank computation is reliable.

In both methods, the common step is the calculation of the ϵ-modes. In the function developed for this module, this step is carried out by using the $compute\_roots\_DDAE$ function explained in the previous section.

Note that, either of the above methods can also be used to determine the μ-CFMs by defining $\nu = 1$ and using $u(t)$ and $y(t)$, defined in (2.27), as the input and the output vector, respectively.

The function developed to find the set of ϵ-CFMs or ϵ-DFMs is named as tds_fm and is called as follows:

$$\gg \quad fms = tds\_fm(ddae, options)$$

where *ddae* is the system object (created, e.g., as explained in Section 5.1) and fms is the resulting set of ϵ-DFMs (or ϵ-CFMs if $\nu = 1$). Here the *options* include all the options of the function $compute\_roots\_DDAE$, as explained in the previous section. The particular options are as follows:

- **method** specifies the method to be used to calculate the fixed modes. When set to $'rt'$ (which is the default) the rank test is used. When set to $'np'$ the numerical procedure is used.

- **root_accuracy_fm** is used only when the **method** is set to $'np'$ and specifies

the maximum allowed difference between eigenvalues in the computational procedure. The default value for this option is 10^{-5}.

5.5. Computation of γ_ψ

In order to compute (2.11), for some $\psi \in \mathbb{R}$, a predictor-corrector approach is used, as it was done in [13]. In the prediction step, Dekker-Brents method is used, where the evaluations are approximated by restricting each θ_i to a grid. The function used to compute γ_ψ is the same function as in TDS_STABIL and is called as

$$\gg \quad gamma\_psi = computegammapsi(diff, psi)$$

where the output of this function is the computed value of γ_ψ and $diff$ can be obtained by calling the function $getdiff$ as

$$\gg \quad diff = getdiff(ddae)$$

where $ddae$ is a system object and $diff$ corresponds to the associated DDE in the form of (2.7).

5.6. Computation of ϵ-blocking zeros

In order to carry out the initialization procedure, given in Section 3.4, real blocking zeros of a centralized system, Σ_s^c, which is obtained from Σ_s by taking u_{s+1} as the only input and y_{s+1} as the only output, with real part greater than or equal to ϵ, must be computed. In order to obtain Σ_s^c, function $getcentralizedtds$ can be called as

$$\gg \quad DDAE\_c = getcentralizedtds(openDDAE, agentno)$$

where $openDDAE$ is a centralized system object and $agentno$ specifies s. Then, the TFM of the system Σ_s^c must be obtained as in (3.30). In order to construct TFM, a MATLAB function tds_evaltf has been developed. This function is called as

$$\gg \quad TFM = tds\_evaltf(DDAE\_c)$$

where *ddae_c* is the centralized system object and TFM is a symbolic representation of (3.30).

Once the TFM is obtained, the ϵ-blocking zeros can be computed by calling

$$\gg \quad bzeros = computebzeros(TFM, region)$$

where TFM is a symbolic representation of (3.30) and *region* is a vector, whose elements specify the boundaries of the region where the first two elements indicate the left and right hand side boundaries, respectively, and the last two elements indicate the lower and upper boundaries, respectively.

The real ϵ-blocking zeros of (3.30) can be calculated by computing the common real ϵ-blocking zeros of the elements of (3.30) which are quasi-polynomials. In order to compute the roots of a quasi-polynomial, we use a MATLAB based *quasi-polynomial mapping based root-finder* (QPmR) algorithm [50]. It should be noted that, if Σ_s^c is a single-input single-output system, then there is only one element of (3.30). However, if Σ_s^c is a multi-input multi-output system, then there will be more than one elements of (3.30). In the latter case, first the real ϵ-blocking zeros of one of the elements of (3.30) are computed. Then, each of these blocking zeros are substituted in the other elements of (3.30) to test whether it is a common blocking zero of all the elements of (3.30), and hence whether it is actually a blocking zero of (3.30).

5.7. Controller Structure

To form the controller matrices in one of the forms described in Section 3.1, a MATLAB function named as *cont_form* has been developed. This function is called as

$$\gg \quad [GFKHtemp, free\_index] = controller\_form(m, q, l)$$

where p, q, and l are the number of output, input, and dimension of the controller, respectively. The function returns $GFKHtemp$, which contains the controller matrices in the form

$$GFKHtemp = \begin{bmatrix} G & F \\ K & H \end{bmatrix} \tag{5.6}$$

where F, G, H, and K are in one of (determined by m, q, and l as explained) the canonical forms (3.14)–(3.17), where all the free parameters are zero. The function also returns $free\_index$, which is a $\tilde{l} \times 2$ dimensional matrix, whose k^{th} row contains the row and column index of the k^{th} free parameter (i.e., the k^{th} element of the parameter vector p of Section 3.1) in the matrix $GFKHtemp$. For example, if the form in (3.14) is used, then the first row of $free\_index$ is $\begin{bmatrix} q+1 & q+1 \end{bmatrix}$, which gives the row and column index of the first free parameter, $a_{1,1}$, in $GFKHtemp$ and the last row of $free\_index$ is $\begin{bmatrix} l+m & l+q \end{bmatrix}$, which gives the row and column index of the last free parameter, $c_{m,l}$, in $GFKHtemp$.

Once $GFKHtemp$ and $free\_index$ are obtained, (5.6) can be formed by calling

$$\gg \quad GFKH = p2GFKH(GFKHtemp, free\_index, p)$$

where p is a free parameter vector, and $GFKHtemp$ and $free\_index$ can be obtained as explained above. Likewise, the free parameters of (5.6) can be formed as a free parameter vector by calling

$$\gg \quad p = GFKH2p(GFKHtemp, free\_index, GFKH)$$

Furthermore, once the closed-loop system object is obtained by using the function *getclosedDDAE* (see Section 5.1), where all the entries of the controller matrices are zero, then, the closed-loop system matrices can be updated by calling

$$\gg \quad closedDDAE\_up = updateclosedDDAE(closedDDAE, GFKH)$$

where $GFKH$ can be obtained bt using the function $p2GFKH$ given above.

5.8. Initialization of Controller Matrices

In order to initialize the controller matrices with respect to the initialization procedure given in Section 3.4, first, the desired eigenvalues of the F matrix must be determined. In this respect, a MATLAB function, named as *computepipmodes*, has been developed. This function is called as

$$\gg \quad dcm = computepipmodes(ddae\_c, options)$$

where *ddae_c* is a centralized system object (which corresponds to Σ_s^c) and *dcm* is the vector which contains the desired initial controller modes. Besides the options given for the other functions of the software, *options* contain the following variables:

- **udcm** is a vector which contains the user-defined initial controller modes. It is an empty vector as a default value.
- **delta** is a non-negative real number which determines the imaginary boundaries of the region $\mathcal{D}$ used in the root finding algorithm QPmR. When this option is set to δ, the lower and upper imaginary boundaries of the region $\mathcal{D}$ are respectively set to $-\delta$ and δ. The default value is 0.1.

The function *computepipmodes* first computes the ϵ-modes of the system using *compute_roots_DDAE* (see Section 5.3), for some $\epsilon < \mu$, which is given by the option **minimal_real_part**.

Note that, if the system is a part of a decentralized system, i.e., it may have ϵ-CFM(s), then, each of ϵ-modes must be checked whether it is a ϵ-CFM (see Section 5.4). By eliminating the fixed modes, only the non-fixed ϵ-modes will be considered, hereafter.

Then, a vector, called *tds_real_modes*, containing the real ϵ-modes is created. If there are any real user-defined controller modes with real part greater than or equal to ϵ, these are also concatenated onto *tds_real_modes*. By treating such user-defined controller modes as a system mode, we avoid to break PIP because of these modes. Then, the transfer function matrix is obtained by using *tds_evaltf* and the real ϵ-blocking zeros located in the region $\mathcal{D}$, with the boundaries $\epsilon \leq \mathrm{Re}(\mathcal{D}) \leq \xi$ and $-\delta \leq \mathrm{Im}(\mathcal{D}) \leq \delta$ are computed by using *computebzeros* (see Section 5.6 above). Here, δ is defined by the option **delta** and ξ is the largest element in *tds_real_modes*. Here, we take into account, not only the real ϵ-blocking zeros, but also complex-conjugate pairs of blocking zeros with a fairly small (i.e., less than or equal to δ) imaginary part. Thus, we treat the real part of such a couple also as a real blocking zero and form the vector *tds_real_zeros* which contains the real blocking zeros and the real parts of the complex-conjugate pairs of blocking zeros in the region $\mathcal{D}$. The elements of *tds_real_zeros* are sorted in an ascending order. The maximum real part of $\mathcal{D}$ is chosen as ξ, since the real zeros to the right of ξ are irrelevant (see [17]). Let

ξ_{rm} denote the rightmost element of *tds_real_zeros*. If there are an odd number of elements in *tds_real_modes* between ξ_{rm} and ξ (including ξ), a real controller mode should be added between ξ_{rm} and the next real blocking zero to the right of it (if there is one) or anywhere to the right of ξ_{rm} (if there are no real blocking zeros to the right of it). In both cases, we can satisfy this requirement by simply adding one real controller mode between ξ_{rm} and ξ.

Let n_{rz} be the number of elements in *tds_real_zeros* and c_i, $i \in \overline{n_{rz}}$, be the i^{th} element of *tds_real_zeros*. Then, for $i = 1, \ldots, n_{rz} - 1$, the software identifies the number of elements of *tds_real_modes* between c_i and c_{i+1}. If there is an odd number of such elements, then a real number in the range $\big(c_i + (c_{i+1} - c_i)/3\ ,\ c_{i+1} - (c_{i+1} - c_i)/3\big)$ is randomly chosen and added to the vector *dcm*. The software also identifies the number of elements of *tds_real_modes* between $c_{n_{rz}}$ and ξ. If this number is odd, then a real number in the range $\big(c_{n_{rz}} + (\xi - c_{n_{rz}})/3\ ,\ \xi - (\xi - c_{n_{rz})}/3\big)$ is randomly chosen and added to the vector *dcm*. Finally, the user-defined initial controller modes, specified by the option **udcm** (if any), are also added to form the vector *dcm*.

Once the desired initial controller modes are determined as above, the function *inital_cont* initializes the controller matrices. This function is called as

$$\gg \quad GFKH = initial\_cont(GFKHtemp, m, q, l, dcm)$$

where $GFKH$ is the controller matrix structured as in (5.6) with all free parameters set to zero (e.g., created by $controller\_form$), m, q, and l are as defined in Section 5.7, and *dcm* is the vector containing the desired initial controller modes (e.g., as determined by *initial_modes*).

If the size of *dcm* (call it l_{dcm}) is less than l, then the function *init_cont* first determines, randomly, $l - l_{dcm}$ number of additional initial controller modes with real part less than ϵ (i.e., the additional initial controller modes are chosen ϵ-stable; see [52] for details). Then, using symbolic and numeric computation, the free parameters of the controller dynamics matrix, F, are determined so that F (which is in one of the canonical forms described in Subsection 3.1) has the desired initial controller modes as its eigenvalues.

Then, the elements of the K matrix are chosen randomly according to a normal distribution centered at zero. It is then checked whether $\gamma_\psi(p^K) < 1$. If this condition is not satisfied, the magnitude of all the elements of K are divided by two until this condition is satisfied.

Finally, the free parameters of H and G are chosen randomly according to a normal distribution centered at zero. Finally, *inital_cont* forms the controller matrix $GFKH$, which is structured as in (5.6), and returns it.

5.9. Graphical User Interface

Once, the system object is created by using the *tds_create* function, all of the features of DCD-TDS can be utilized by using the Graphical-User-Interface (GUI) which employs the developed functions given in the previous sections. The associated GUI of DCD-TDS is developed in [53] and introduced in [54]. Furthermore an illustration of this tool can be found in [35].

There are two separate GUI panels where the first one, which is shown in Figure 5.1., is used to analyze the spectral characteristic of a given decentralized (or centralized) time-delay system, and the second one, which is shown in Figure 5.2., is used to manage the design of a decentralized (or centralized) controller in the form of (2.30) (or (3.3)).

In the next two subsections, for a given time-delay system, analysis and design facilities of the developed GUI panels are introduced.

5.9.1. Analysis Phase

The GUI panel used for the analysis phase, which is shown in Figure 5.1., has several sub-panels so that each one can be used to carry out different tasks, separately. For each sub-panel, an input (i.e., system object) must be loaded, beforehand. At the end, the output, which may be a system object or a variable with an appropriate data type, is exported to the MATLAB Workspace.

Besides the modules of the GUI, which are introduced in the previous sections, there is also a visualization tool embedded. By using this tool, user can obtain plots (e.g., the plots shown in Chapter 6) of modes and/or zeros of any

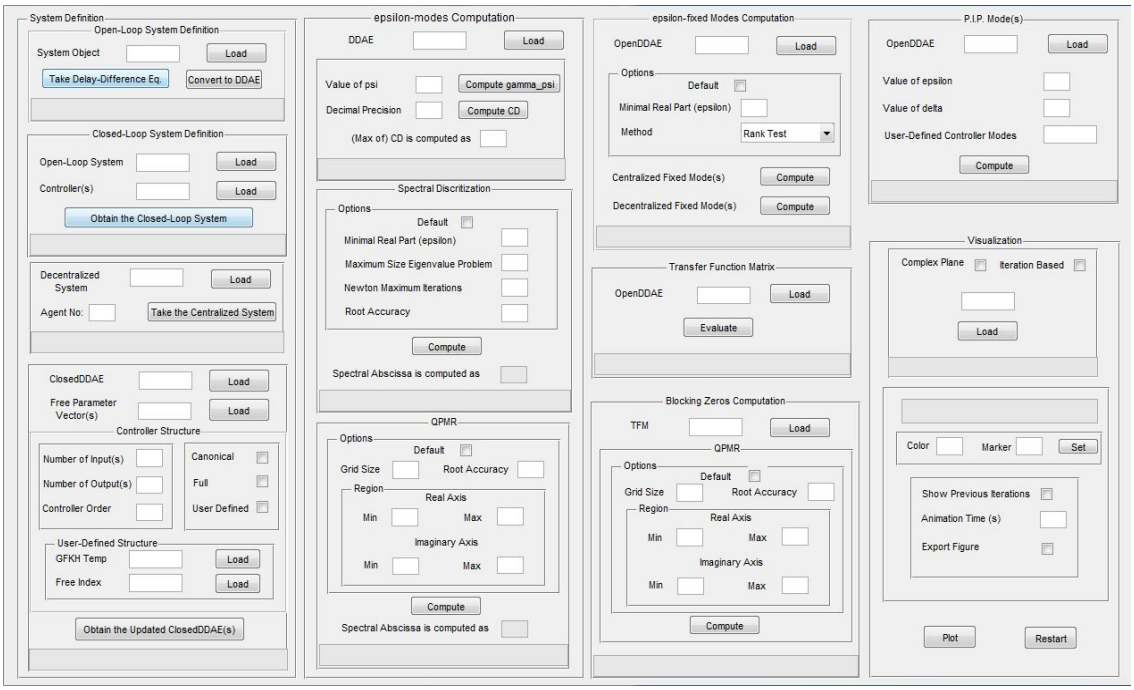

Figure 5.1. Analysis panel of the GUI

system. Furthermore, animated plots, which monitor the evolutions of spectral abscissa, safe upper bound, system modes, and the controller modes during the process of optimization can also be displayed.

5.9.2. Design Phase

The design phase of the GUI panel is shown in Figure 5.2.. This panel has three main sub-panels. The user should begin with entering a decentralized or centralized system. Once the system object is loaded, program will recognize whether it is centralized or decentralized. If the system is centralized, then the decentralized controller design sub-panel will be blocked out. However, if the system is decentralized, then, the first sub-panel is used. We begin with uploading the object of the system, which is in the form of (5.2), obtained by using the function $tds2DDAE$.

In the decentralized controller design algorithm, one of the important points is the design order for the control agents. Since, there is no structural restriction on the design order of each control agent, it can be the same as the order of the agents as defined in the system object. However, empirical evidence has shown that, it may be better to use a different design order to obtain lower dimensional

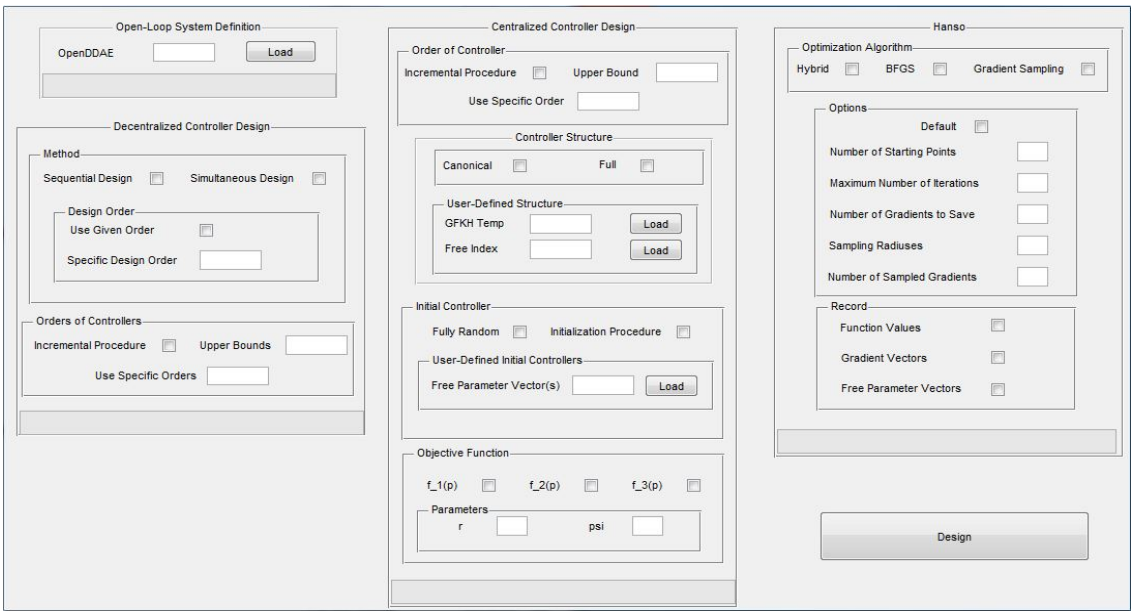

Figure 5.2. Design panel of the GUI

controllers. Thus, the program allows the user to provide a design order by entering a $1 \times \nu$-dimensional vector.

As it is stated in the Chapter 4, the decentralized controller design algorithm starts with the minimum possible controller dimension, which is determined at the beginning, and increases the dimension when the controller with current dimension fails. However, increasing the dimension of the lower-indexed controllers may sometimes be unnecessary, since those modes which can not be stabilized by a low order lower-indexed controller can be stabilized at the further stages of the algorithm. Therefore, the software also offers the option to define upper bounds for the dimension of each controller by entering a $1 \times \nu$-dimensional vector. Alternatively, user can also design local controllers with specified orders by entering a $1 \times \nu$-dimensional vector.

The controller design algorithm also allows to predefine the controller structure. Thus, some entries of each controller matrices can be defined as fixed parameters. By this way, only the free parameters will be tuned during the optimization. Here, the program offers three options. The first one is to let all the entries of the controller matrices be free parameters. The second option is to define a specific controller structure. To use this option, however, the controller dimensions must also be prespecified by the user. The third option, which we find most useful as

stated in Chapter 3, is to use one of the canonical forms given in Section 3.1.

Once the controller structure is defined, the initial controller (i.e., initial values of the free parameters) must be determined. These parameters can be chosen randomly according to a normal distribution, or can be specified by the user, or the initialization procedure described in Section 3.4 and associated functions introduced in Section 5.8 can be used.

As a final step, user should specify the options of the optimization package HANSO. As it was mentioned in Section 2.2, HANSO has two main optimization algorithms, GS and BFGS, and a hybrid code which uses the main algorithms consecutively. In fact, there are several options for parameters of the optimization algorithms. Therefore, it is advised to use the default options.

6. EXAMPLES

6.1. Centralized Controller Design

Let us consider a LTI neutral time-delay system, described in the form of (3.2) as

$$\begin{aligned} &\dot{\tilde{x}}(t) + \tilde{E}_3\dot{\tilde{x}}(t-0.7) + \tilde{E}_4\dot{\tilde{x}}(t-1.7) = \tilde{A}_0\tilde{x}(t) + \tilde{B}_2u(t-0.5) \\ &y(t) = \tilde{C}_1\tilde{x}(t-0.1) + 2.4u(t-2) \end{aligned} \tag{6.1}$$

where

$$\tilde{E}_3 = \begin{bmatrix} 0 & -0.2 & 0.4 \\ 0.5 & -0.3 & 0 \\ -0.2 & -0.7 & 0 \end{bmatrix}, \quad \tilde{E}_4 = \begin{bmatrix} 0.3 & 0.1 & 0 \\ 0 & -0.2 & 0 \\ -0.1 & 0 & -0.4 \end{bmatrix},$$

$$\tilde{A}_0 = \begin{bmatrix} -4.8 & 4.7 & 3 \\ 0.1 & 1.4 & -0.4 \\ 0.7 & 3.1 & -1.5 \end{bmatrix},$$

$$\tilde{B}_2 = \begin{bmatrix} 0.3 & 0.7 & 0.1 \end{bmatrix}^T, \quad \text{and} \quad \tilde{C}_1 = \begin{bmatrix} 0.5 & -0.8 & 0.01 \end{bmatrix},$$

which is obtained by modifying the example in [9] (since state vector feedback, rather than output feedback, was considered in [9], the example in [9] has only dynamic equations, therefore, an arbitrary output equation is added).

In order to create the system object, we use the function *tds_create* which is introduced in Section 5.1. Before calling this function, some properties of the system must be predefined as

$$\gg metadata.Nagent = 1;$$
$$\gg metadata.systype = 'neutral';$$

Now, the system object can be created as

$$\gg \quad tds\_c = tds\_create(\{eye(3), \tilde{E}_3, \tilde{E}_4\}, [0, 0.7, 1.7], \{\tilde{A}_0\}, 0, \{\tilde{B}_2\}, 0.5, \{\tilde{C}_1\}, 0.1, 2.4, 2, metadata);$$

Once, the system object is created, analysis and controller design features of DCD-TDS can be utilized by using the GUI panels given in Section 5.9.

To analyze the spectral properties of the system (6.1), the associated system object in the form of (3.1) must first be obtained by using the first sub-panel which uses the function $tds2DDAE$. By defining

$$\delta(t) := \tilde{x}(t) + \tilde{E}_3\tilde{x}(t-0.7) + \tilde{E}_4\tilde{x}(t-1.7)$$

and $x(t) := \begin{bmatrix} \delta(t)^T & \tilde{x}(t)^T \end{bmatrix}^T$, the system (6.1) can be rewritten in the form of (3.1), where $h_1 = 0.1$, $h_2 = 0.5$, $h_3 = 0.7$, $h_4 = 1.7$, $h_5 = 2$,

$$E = \begin{bmatrix} I_3 & 0 \\ 0 & 0_{3\times3} \end{bmatrix},\ A_0 = \begin{bmatrix} 0 & \tilde{A}_0 \\ -I_3 & I_3 \end{bmatrix},\ A_3 = \begin{bmatrix} 0_{3\times3} & 0 \\ 0 & \tilde{E}_3 \end{bmatrix},\ A_4 = \begin{bmatrix} 0_{3\times3} & 0 \\ 0 & \tilde{E}_4 \end{bmatrix},$$

$$B_2 = \begin{bmatrix} \tilde{B}_2^T & 0_{1\times3} \end{bmatrix}^T,\quad C_1 = \begin{bmatrix} 0_{1\times3} & \tilde{C}_1 \end{bmatrix},\quad D_5 = 1\ ,$$

and all other matrices being zero. Let us denote this system by Σ^c. From hereafter, the new system object, which corresponds to Σ^c, will be considered.

Our aim is to design a controller of the form (3.3) to strongly μ-stabilize Σ^c, for $\mu = 0$. We also aim $C_D(\Sigma^{cl}) < \psi$ for $\psi = -0.1$.

We calculate $C_D(\Sigma^c) = -0.2751$ and $\gamma_{-0.1}(\Sigma^c) = 0.8312$. Thus, we can calculate $\Omega_\varphi(\Sigma^c)$ by the method of [49], for any $\varphi > -0.2751$. We choose $\varphi = -0.25$ and determine $\Omega_{-0.25}(\Sigma) = \{0.2180, 0.0976 \pm 1.0396\mathrm{j}\}$. Hence, the given system is not 0-stable, since there exist 0-modes at $s_1 = 0.2180$ and $s_{2,3} = 0.0976 \pm 1.0396\mathrm{j}$. As it is emphasized in Section 5.3, there are finitely many $\bar{\varphi}$-modes inside any finite region. Relying on that, we choose $\bar{\varphi} = -2$. Then, by using the QPmR tool of [50], we calculate all the -2-modes of Σ^c with imaginary part between -50 and 50 and plot those in Figure 6.1. using red stars. Also, we calculate the roots of the DDE inside the same region, which are also shown in Figure 6.1. using blue pluses.

Next, we consider the region $\mathcal{D} := \{s \mid \epsilon \leq \mathrm{Re}(s) \leq \xi = 0.2180\ ,\ -\delta \leq \mathrm{Im}(s) \leq \delta\}$, where we choose $\epsilon = -0.05$ and $\delta = 0.1$, and, using function *computebzeros*, we determine that there is a blocking zero, $\lambda_1 = 0.0413$, which is shown in Figure 6.1. using blue circles, inside $\mathcal{D}$. Since there is one real mode of Σ^c, $s_1 = 0.2180$, located to the right hand side of λ_1, a real controller mode is needed anywhere to the right of that zero. In this regard, the minimum dimension of the controller is predetermined as $l^{\min} = 1$.

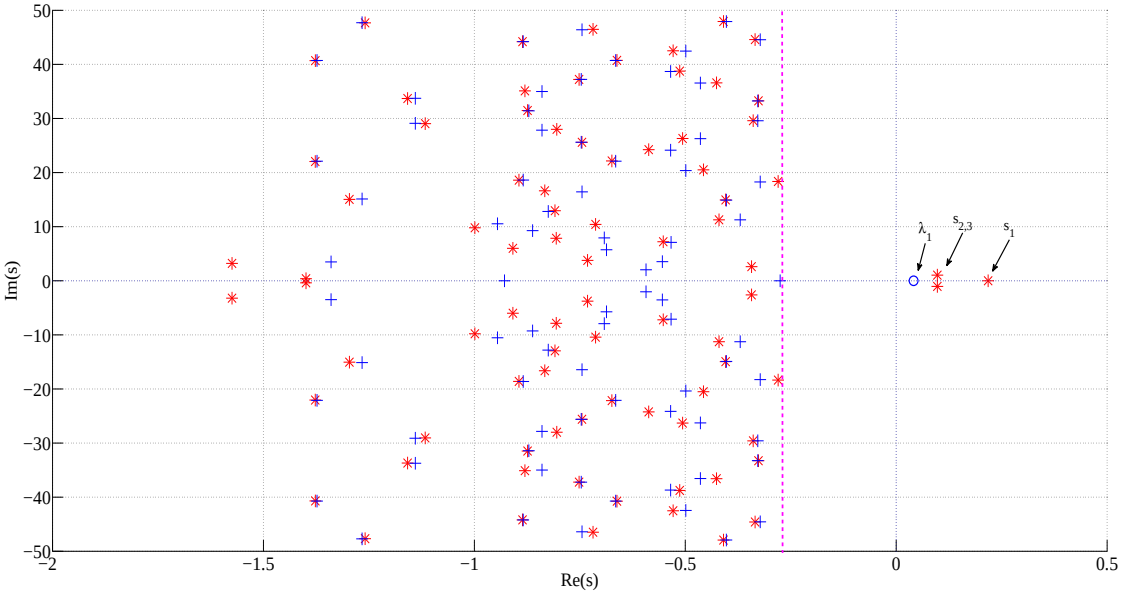

Figure 6.1.−2-modes of Σ^c (red stars), −2-modes of the associated DDE of Σ^c (blue pluses), blocking zero of Σ^c (blue circle), and $C_D(\Sigma^c)$ (magenta dashed line).

According to the initialization procedure, the initial controller mode is chosen as $s^* = 0.1448$. In here, the controller is structured as in (3.14), i.e., controllable canonical form. Thus, $G \equiv 1$ and the initial H value is chosen as 0.7269, randomly according to a normal distribution centered at zero. Then, the initial value of K is chosen as 0.063 which ensures $\gamma_{-0.1}(p^K) = 0.8312 < 1$. Then, starting with the initial controller and by using the BFGS algorithm, given in Section 2.2.2, the program solves (3.18) with $r = 0.001$. Hence, the controller

$$\dot{z}(t) = 0.1332z(t) + y(t)$$
$$u(t) = -0.2097z(t) - 0.0621y(t)$$

which strongly 0-stabilizes the system Σ^c, is obtained. The rightmost modes of the closed-loop system are calculated as -0.0077 ± 0.8222j, thus, $c(\Sigma^{cl}) = -0.0077$. We calculate $\gamma_{-0.1}(\Sigma^{cl}) = 0.8312$ and $C_D(\Sigma^{cl}) = -0.2751$. The −2-modes of both the closed-loop system and the associated DDE with imaginary part between −50 and 50 are shown in Figure 6.2..

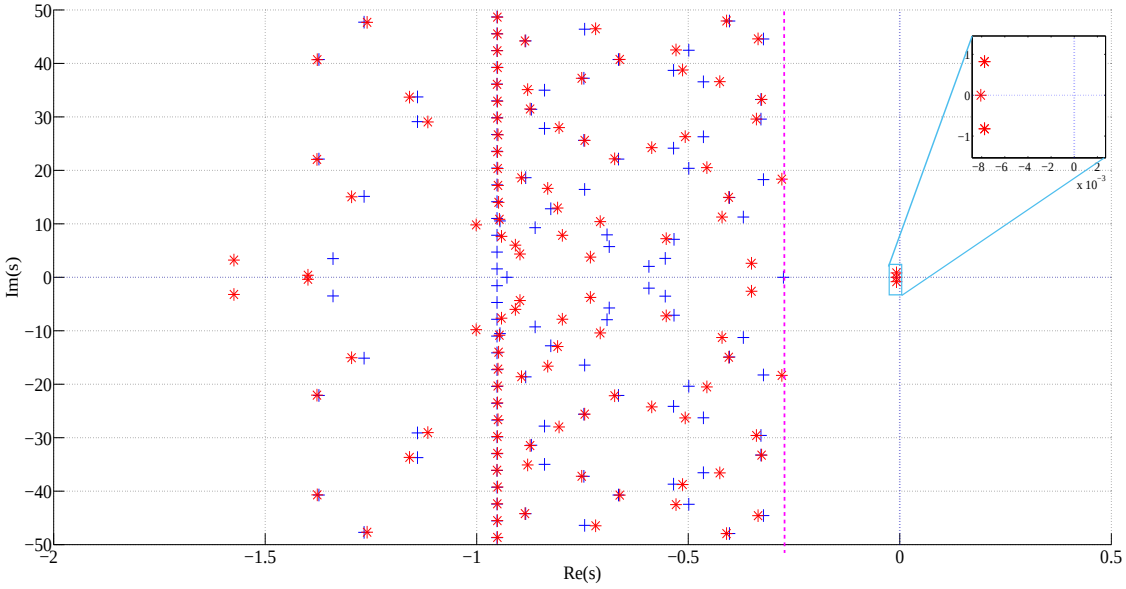

Figure 6.2.−2-modes of Σ^{cl} (red stars), −2-modes of the associated DDE of Σ^{cl} (blue pluses), and $C_D(\Sigma^{cl})$ (magenta dashed line).

6.2. Decentralized Controller Design

Let us consider a LTI decentralized retarded time-delay system, described in the form of (2.24) as

$$\begin{aligned}\dot{x}(t) &= A_0x(t) + A_1x(t-1) + B_{1,0}u_1(t) + B_{1,1}u_1(t-1)\\ &\qquad + B_{2,0}u_2(t) + B_{2,1}u_2(t-1)\\ y_1(t) &= C_{1,0}x(t) + C_{1,1}x(t-1) + u_1(t-2)\\ y_2(t) &= C_{2,0}x(t) + C_{2,1}x(t-1) - u_2(t-2)\end{aligned} \tag{6.2}$$

where

$$A_0 = \begin{bmatrix} 7 & 9 & 7 & 9\\ 0 & -1 & 4 & -2\\ -11 & -6 & -7 & -11\\ -22 & -12 & -4 & -27\end{bmatrix}, \quad A_1 = \begin{bmatrix} 4 & 6 & -8 & -1\\ 0 & 4 & 0 & 0\\ 5 & -3 & 9 & 1\\ 10 & -6 & 6 & 8\end{bmatrix},$$

$$B_{1,0} = \begin{bmatrix} -4\\ -3\\ 2\\ 4\end{bmatrix}, \quad B_{1,1} = \begin{bmatrix} 2\\ 1\\ -1\\ -2\end{bmatrix}, B_{2,0} = \begin{bmatrix} 3\\ 0\\ -3\\ -5\end{bmatrix}, B_{2,1} = \begin{bmatrix} 1\\ 0\\ -1\\ -1\end{bmatrix},$$

$$C_{1,0} = \begin{bmatrix} 0 & 1 & 4 & -2\end{bmatrix}, \quad C_{1,1} = \begin{bmatrix} 1 & -1 & 1 & 0\end{bmatrix},$$

$$C_{2,0} = \begin{bmatrix} 1 & -1 & 2 & -0.5 \end{bmatrix} \ , \ C_{2,1} = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \ .$$

This example is obtained by modifying the example in [55]. Here, the delayed direct feedthrough terms are added so that, although the given system is retarded, the closed-loop system becomes neutral. The original example is, first, solved by using continuous pole placement algorithm in [55], and then, it is solved by using nonsmooth optimization based fixed-order controller design method in [22].

In order to create the decentralized system object, we use the function *tds_create*. Before calling this function, some properties of the system must be predefined as

$$\gg metadata.Nagent = 2;$$
$$\gg metadata.systype = 'retarded';$$

where the first field of *metadata* specifies the number of control agents (i.e., ν) and the second one is used as an identifier of the system type. Now, the decentralized system object can be created as

$$>> \quad tds = tds\_create(\{A_0, A_1\}, [0,1], \{B_{1,0}, B_{1,1}\}, [0,1], \{B_{2,0}, B_{2,1}\}, [0,1],$$
$$\{C_{1,0}, C_{1,1}\}, [0,1], \{C_{2,0}, C_{2,1}\}, [0,1], \{1\}, 2, \{\}, \{\}, \{-1\}, 2, metadata);$$

where the two empty fields correspond to $D_{1,2,i}$ and $D_{2,1,i}$, which do not exist in our example. Now, analysis and controller design features of DCD-TDS can be utilized by using the GUI panels.

Let Σ_0 indicate the system (6.2). By specifying the control agent numbers, we can obtain the corresponding centralized system objects, which will be denoted by Σ_0^{c1} and Σ_0^{c2}, which have only $\{u_1, y_1\}$ and $\{u_2, y_2\}$ as input-output pairs, respectively.

Our aim is to design a decentralized controller of the form (2.30) to μ-stabilize Σ_0, for $\mu = -0.1$. However, even though Σ_0 is retarded type, because of the delayed direct feed-through terms, Σ_1 and Σ_2 will be neutral type. Therefore, our aim should be extended to design a strongly μ-stabilizing decentralized controller. Thus, we also aim $C_D(\Sigma_2) < \psi$ for $\psi = -0.3$.

Since, Σ_0 is retarded type, thus, $C_D(\Sigma_0) = -\infty$, we can calculate $\Omega_\varphi(\Sigma_0)$ by the method of [49], for any finite φ. We choose $\varphi = -0.2$ and determine

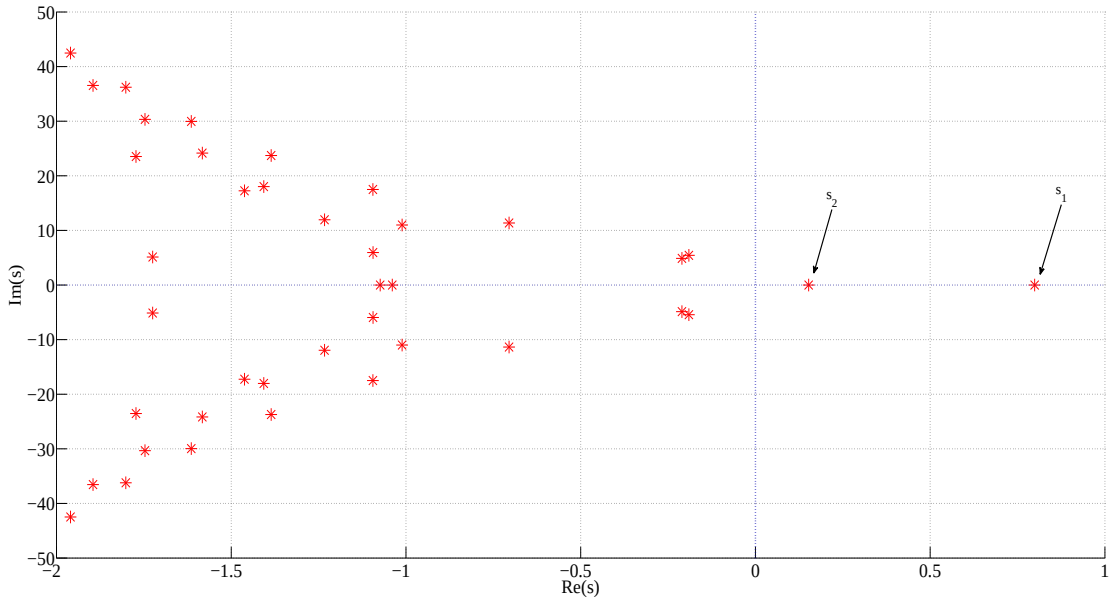

Figure 6.3.−2-modes of Σ_0 (red stars).

$\Omega_{-0.2}(\Sigma) = \{0.7990, 0.1523, -0.1904 \pm 5.4367\mathrm{j}\}$. Hence, the given system is not -0.1-stable, since there exist -0.1-modes at $s_1 = 0.7990$ and $s_2 = 0.1523$. We also calculate all the -2-modes of Σ_0 and plot those in Figure 6.3. using red stars.

Furthermore, we determine that s_2 is a -0.1-CFM for control agent 1, and s_1 is a -0.1-CFM for control agent 2. Hence, the system is not -0.1-stabilizable by only one control agent. However, we see that there are no -0.1-DFMs. Hence, it is possible to design a strongly -0.1-stabilizing decentralized controller for Σ_0.

Now, we can perform the controller design. First, we choose the design order as defined in the system object, thus, we enter $design\_order = [1, 2]$. Furthermore, we define the upper bounds for the dimension of each controller by entering $upperbounds = [3, 3]$. Also, we choose to design each controller in a suitable canonical form (see Section 3.1).

First, a controller for the first control agent is designed. Since, there exist no -0.2-blocking zeros in the region $\mathcal{D} := \{s \mid \epsilon = -0.2 \leq \mathrm{Re}(s) \leq \xi = 0.7990\ ,\ -\delta \leq \mathrm{Im}(s) \leq \delta\}$, where we choose $\delta = 0.1$, there is no need to choose a particular initial controller mode. Therefore, we let $l_1^{min} = 0$. However, the algorithm fails to design a strongly -0.1-stabilizing controller with dimension $l_1 = 0$ and $l_1 = 1$. Finally, by

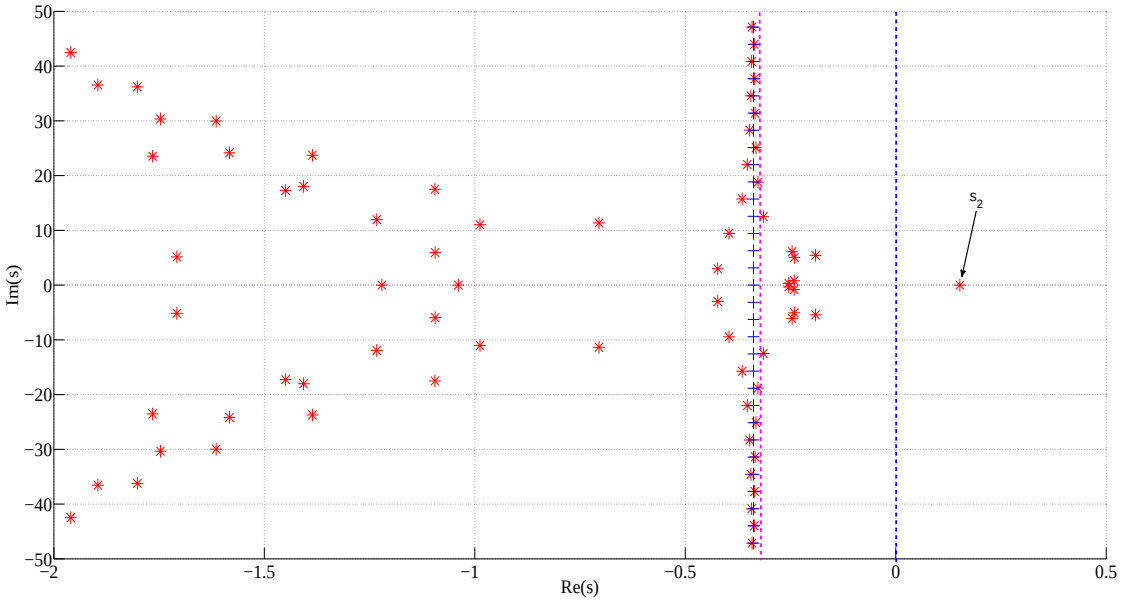

Figure 6.4.−2-modes of Σ_1 (red stars), -2-modes of the associated DDE of Σ_1 (blue pluses), and $C_D(\Sigma_1)$ (magenta dashed line).

choosing the initial controller modes as $s^*_{1,2} = -0.6140 \pm 1.3666\text{j}$, the controller

$$\dot{z}_1(t) = \begin{bmatrix} 0 & 1 \\ -1.3862 & 0.0198 \end{bmatrix} z_1(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} y_1(t)$$
$$u_1(t) = \begin{bmatrix} 0.0954 & 0.8849 \end{bmatrix} z_1(t) + 0.5087 y_1(t)$$

which strongly -0.1-stabilizes the system Σ_0^c (except for its CFM), is obtained. By applying this controller to Σ_0 the system Σ_1 is obtained. The -2-modes of Σ_1 and its associated DDE are shown in Figure 6.4. with red stars and blue pluses, respectively. It is seen that the only unstable mode of Σ_1 is $s_2 = 0.1523$, which was the CFM of Σ_0^c. Furthermore, it is calculated that $\bar{c}(\Sigma_1) = -0.1904$, $C_D(\Sigma_1) = -0.3379$, and $\gamma_{-0.3}(\Sigma_1) = 0.9269$.

Since, there is no -2-blocking zeros of Σ_1 in the region $\mathcal{D} := \{s \mid \epsilon = -0.2 \leq \text{Re}(s) \leq \xi = 0.1523\ ,\ -0.1 \leq \text{Im}(s) \leq 0.1\}$, we let $l^{min} = 0$. However, the algorithm fails to design a strongly -0.1-stabilizing controller for the second control agent, with dimension $l_2 = 0$, $l_2 = 1$, and $l_2 = 2$. However, with $l_2 = 3$, the

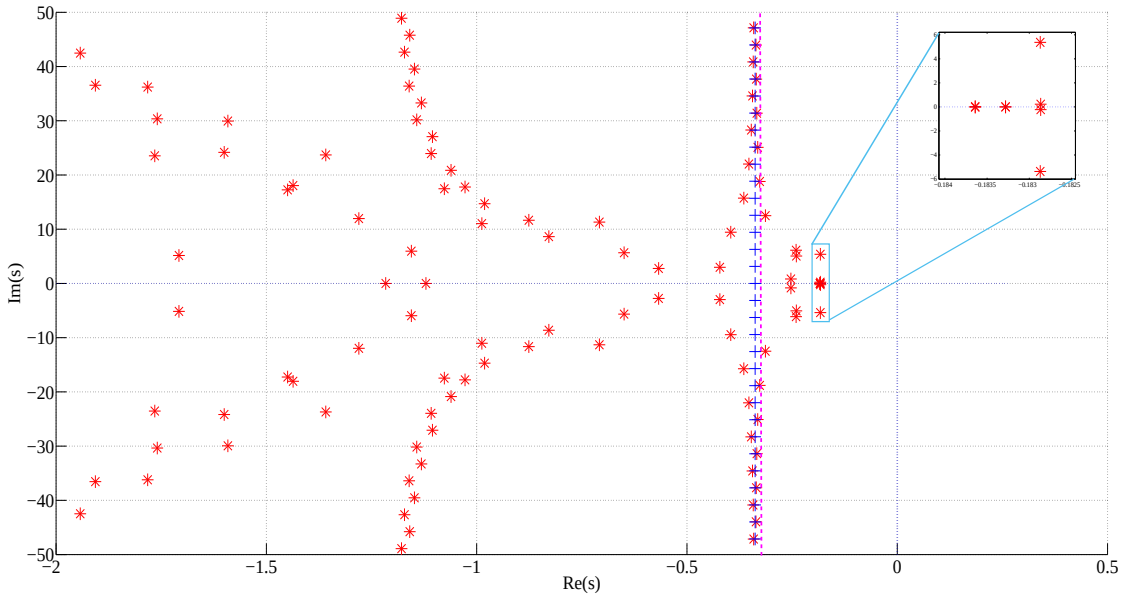

Figure 6.5.−2-modes of Σ_2 (red stars), −2-modes of the associated DDE of Σ_2 (blue pluses), and $C_D(\Sigma_2)$ (magenta dashed line).

controller

$$\dot{z}_2(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -0.9732 & -1.88 & -4.9083 \end{bmatrix} z_2(t) + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} y_2(t)$$

$$u_2(t) = \begin{bmatrix} -0.3123 & -0.8292 & -1.9493 \end{bmatrix} z_2(t) + 0.0909 y_2(t)$$

which strongly -0.1-stabilizes the system Σ_1^c, is obtained. By applying this controller to Σ_1 the system Σ_2 is obtained. It is calculated that $c(\Sigma_2) = -0.1829$. The -2-modes of Σ_2 and its associated DDE are shown in Figure 6.5. with red stars and blue pluses, respectively. Furthermore, strong -0.1-stability is also achieved, since, $C_D(\Sigma_2) = -0.3379$ and $\gamma_{-0.3}(\Sigma_2) = 0.9269$.

7. CONCLUSION

In this thesis, both centralized and decentralized controller design for LTI time-delay systems has been considered. The main objective was to design a finite dimensional centralized/decentralized output feedback controller which strongly μ-stabilizes the centralized/decentralized TDS, for a given stability and strong stability boundaries μ and ψ, respectively.

In this respect, first attempt has been made to design a strongly μ-stabilizing centralized output feedback controller for a given centralized TDS whose autonomous part can be described by DDAEs. In order to design such a controller, a nonsmooth optimization based fixed-order controller design method, introduced in [13], has been used. Since, this method is eigenvalue-based, the stability conditions, given in [4], have been considered. The proposed method of [13] have been enhanced by making two main contributions. First, the controllers are structured in certain canonical forms. By this way, the number of free parameters of the controller matrices are reduced, so, the computational burden of the optimization is reduced. Second, considering the nonconvexity of the optimization problem, an initialization procedure has been proposed as a novel contribution of this thesis to the nonsmooth optimization based fixed-order controller design method for both finite-dimensional and infinite dimensional systems. By choosing the most favorable initial controller, the convergence of the optimization algorithm has been facilitated.

In order to design a decentralized controller for a LTI time-delay system, the stabilizability conditions of decentralized LTI time-delay systems, given in [26], has been considered. In this respect, a design algorithm, which utilizes decentralized pole assignment algorithm of Davison and Chang [34], has been proposed. In the proposed algorithm, a finite-dimensional output feedback dynamic controller was designed by using the proposed nonsmooth optimization based centralized controller design algorithm, for each control agent, sequentially. One of the important points in this algorithm was the controller design order of the control agents. This design order may be the same as the order of the control agents as defined beforehand. However, it may be better to use a different design order to obtain lower dimensional controllers and/or to move all the modes further to the left. Unfortunately, there

is no any certain rule to decide which design order is better. In this respect, the developed software was extended so that controllers are designed with respect to a given order by the user. One way to achieve designing controllers according to a given order is to change the definition of the given system object so that in the new object the order of the control agents is same as the desired order for controller design.

Furthermore, a new software package DCD-TDS, which makes use of the slightly customized version of related functions in the software package TDS_STABIL [13] has been developed. DCD-TDS enables to analyze both centralized and decentralized TDSs. Furthermore, DCD-TDS is able to design both centralized and decentralized controllers by using the developed controller design algorithms. In the controller design part, the MATLAB based software package named HANSO is employed to solve nonsmooth optimization problems. Considering the complexity of the software, a GUI, which provides an efficient utilization, has been developed.

As a possible future work, infinite-dimensional, besides finite-dimensional, centralized/decentralized controller design for centralized/decentralized TDSs can be considered. It was proved in [26] that, provided that $\mu > c_D$, the system Σ (and Σ^c) can be μ-stabilized by a LTI time-delay output feedback controller if and only if it can be $^-\mu$-stabilized by a LTI finite-dimensional output feedback controller. It is well-known that a very high dimensional controller may be needed to μ-stabilize a TDS, if the controllers are restricted to be finite-dimensional. Therefore, designing a time-delay controller, rather than a finite-dimensional controller, may be advantageous. Furthermore, DCD-TDS is still open for improvement. Therefore, DCD-TDS can be enhanced in the near future. Moreover, DCD-TDS can be extended in the light of the above mentioned future works.

REFERENCES

[1] R. Sipahi, S. I. Niculescu, C. T. Abdallah, W. Michiels, and K. Gu. Stability and stabilization of systems with time delay. *Control Systems, IEEE*, 31(1):38–65, 2011.

[2] S.-I. Niculescu. *Delay Effects on Stability: A Robust Control Approach,* Lecture Notes in Control and Information Sciences, No. 269. Springer-Verlag, London, 2001.

[3] J.-P. Richard. Time-delay systems: An overview of some recent advances and open problems. *Automatica*, 39:1667–1694, 2003.

[4] W. Michiels and S. I. Niculescu. *Stability and Stabilization of Time-Delay Systems*. SIAM, Philadelphia, PA, 2007.

[5] J.K. Hale and S.M. Verduyn-Lunel. *Introduction to functional differential equations*. Springer, New York, USA, 1993.

[6] J. J. Loiseau, M. Cardelli, and X. Dusser. Neutral-type time-delay systems that are not formally stable are not BIBO stabilizable. *IMA Journal of Mathematical Control and Information*, 19:217–227, 2002.

[7] W. Michiels and S.-I. Niculescu. *Stability, Control, and Computation for Time-Delay Systems: An Eigenvalue-Based Approach*. SIAM, Philadelphia, PA, 2014.

[8] W. Michiels, K. Engelborghs, P. Vansevenant, and D. Roose. Continuous pole placement for delay equations. *Automatica*, 38(5):747 – 761, 2002.

[9] W. Michiels and T. Vyhlidal. An eigenvalue based approach to the robust stabilization of linear time delay systems of neutral type. *Automatica*, 41:991–998, 2005.

[10] J. V. Burke, D. Henrion, A. S. Lewis, and M. L. Overton. Stabilization via nonsmooth, nonconvex optimization. *IEEE Transactions on Automatic Control*, 51:1760–1769, 2006.

[11] J. Vanbiervliet, K. Verheyden, W. Michiels, and S. Vandewalle. A nonsmooth optimization approach for the stabilization of time-delay systems. *ESAIM Control, Optimisation, and Calculus of Variations*, 14:478–493, 2008.

[12] T. Vyhlidal, W. Michiels, and P. Mcgahan. Synthesis of strongly stable state-

derivative controllers for a time-delay system using constrained non-smooth optimization. *IMA Journal of Mathematical Control and Information*, 27:437–455, 2011.

[13] W. Michiels. Spectrum based stability analysis and stabilisation of systems described by delay differential algebraic equations. *IET Control Theory and Applications*, 5:1829–1842, 2011.

[14] S. Gumussoy and W. Michiels. Fixed-order H-infinity control for interconnected systems using delay differential algebraic equations. *SIAM Journal on Control and Optimization*, 49:2212–2238, 2011.

[15] W. Michiels, T. Vyhlidal, and P. Zitek. Control design for time-delay systems based on quasi-direct pole placement. 27:337–343, 2010.

[16] D. Pilbauer, T. Vyhlidal, and W. Michiels. Spectral design of output feedback controllers for systems pre-compensated by input shapers. In *Preprints of the 12th IFAC Workshop on Time-delay Systems*, Ann Arbor, MI, USA, June 2015.

[17] S. M. Özer and A. İftar. Controller design for neutral time-delay systems by nonsmooth optimization. In *Preprints of the 13th IFAC Workshop on Time-delay Systems*, Istanbul, Turkey, June 2016.

[18] A. S. Lewis. Nonsmooth optimization and robust control. *Annual Reviews in Control*, 31(2):167 – 177, 2007.

[19] J. V. Burke, A. S. Lewis, and M. L. Overton. A robust gradient sampling algorithm for nonsmooth, nonconvex optimization. *SIAM Journal on Optimization*, 15(3):751–779, 2005.

[20] A. S. Lewis and M. L. Overton. Nonsmooth optimization via BFGS. *SIAM Journal on Optimization*, 2009. Submitted.

[21] M. L. Overton. HANSO: A hybrid algorithm for nonsmooth optimization. Technical report. http://cs.nyu.edu/overton/software/hanso.

[22] S. M. Özer and A.İftar. Decentralized controller design for time-delay systems by optimization. In *Preprints of the 12th IFAC Workshop on Time-delay Systems*, Ann Arbor, MI, USA, June 2015.

[23] D. D. Šiljak. *Decentralized Control of Complex Systems.* Academic Press, San Diego, 1991.

[24] J. Lunze. *Feedback Control of Large Scale Systems.* Prentice Hall, New York,

1992.

[25] L. Bakule. Decentralized control: An overview. *Annual Reviews in Control*, 32:87–98, 2008.

[26] H. E. Erol and A. İftar. Stabilization of decentralized descriptor-type neutral time-delay systems by time-delay controllers. *Automatica*, 64:262–269, 2016.

[27] M. S. Mahmoud. Decentralized stabilization of interconnected systems with time-variyng delays. *IEEE Transactions on Automatic Control*, 54:2663–2668, 2009.

[28] H. E. Erol. Decentralized control of time-delay systems. Master's thesis, Anadolu University, Eskişehir, Turkey, January 2014.

[29] S. H. Wang and E. J. Davison. On the stabilization of decentralized control systems. *IEEE Transactions on Automatic Control*, 18:473–478, 1973.

[30] E. Kamen, P. Khargonekar, and A. Tannenbaum. Stabilization of time-delay systems using finite-dimensional compensators. *IEEE Transactions on Automatic Control*, 30:75–78, 1985.

[31] A. Momeni and A. G. Aghdam. A necessary and sufficient condtion for stabilization of decentralized time-delay systems with commensurate delays. In *Proceedings of the 47th IEEE Conference on Decision and Control*, pages 5022–5029, Cancun, Mexico, 2008.

[32] A. Momeni, A. G. Aghdam, and E. J. Davison. Decentralized fixed modes for LTI time-delay systems. In *Proceedings of the American Control Conference*, pages 6593–6599, Baltimore, USA, 2010.

[33] H. E. Erol and A. İftar. A necessary and sufficient condition for the stabilization of decentralized time-delay systems by time-delay controllers. In *Preprints of the 13th IFAC Symposium on Large Scale Complex Systems*, pages 62–67, Shanghai, China, July 2013.

[34] E. J. Davison and T. N. Chang. Decentralized stabilization and pole assignment for general proper systems. *IEEE Transactions on Automatic Control*, 35:6:652–664, 1990.

[35] S. M. Özer, G. Gülmez, and A. İftar. A software to design decentralized controllers for time-delay systems. In *Proceedings of the IEEE Conference on Computer Aided Control System Design*, Buenos Aires, Argentina, September

2016. To appear.

[36] F. Khorrami, S. Tien, and Ü. Özgüner. DOLORES: a software package for analysis and design of optimal decentralized control. In *Proceedings of the 40th National Aerospace and Electronics Conf.*, pages 434–441, Dayton, USA, May 1988.

[37] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006.

[38] J. V. Burke, A. S. Lewis, and M. L. Overton. Approximating subdifferentials by random sampling of gradients. *Mathematics of Operations Research*, 27:567–584, 2002.

[39] K. C. Kiwiel. Convergence of the gradient sampling algorithm for nonsmooth nonconvex optimization. *SIAM Journal on Optimization*, 18(2):379–388, 2007.

[40] C. G. Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

[41] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[42] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[43] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–656, 1970.

[44] A. S. Lewis and M. L. Overton. Nonsmooth optimization via quasi-newton methods. *Mathematical Programming*, 141(1):135–163, 2012.

[45] A. Skajaa. Limited memory BFGS for nonsmooth optimization. Master's thesis, New York University, January 2010.

[46] J.K. Hale and S.M. Verduyn-Lunel. Effects of small delays on stability and control. In Ran A. C. M. Bart, H. and I. Gohberg, editors, *Operator Theory and Analysis: The M.A. Kaashoek Anniversary Volume Workshop in Amsterdam, November 12–14, 1997*, pages 275–301. Birkhäuser Basel, Basel, 2001.

[47] M. Vidyasagar. *Control System synthesis: A Factorization approach*. M.I.T. Press, Cambridge, MA, 1985.

[48] A. Momeni and A. G. Aghdam. On the stabilization of decentralized time-delay systems. Technical report, Conordia University, Montréal, Canada, 2008.

http://www.ece.concordia .ca/˜aghdam/TechnicalReports/techrep2008_1.pdf. (Accessed November 2012).

[49] Z. Wu and W. Michiels. Reliably computing all characteristic roots of delay differential equations in a given right half plane using a spectral method. *Journal of Computational and Applied Mathematics*, 236:2499–2514, 2012.

[50] T. Vyhlidal and P. Zitek. QPmR - Quasi-polynomial root-finder: Algorithm update and examples. In Lafay J. F. Vyhlidal T. and Sipahi R., editors, *Delay Systems: From Theory to Numerics and Applications*, chapter 10, pages 299–312. Springer, New York, 2014.

[51] T. Vyhlidal and P. Zitek. Mapping based algorithm for large-scale computation of quasi-polynomial zeros. *IEEE Transactions on Automatic Control*, 54(1):171–177, 2009.

[52] S. M. Özer and A. İftar. A software to design decentralized controllers for time-delay systems. In *TOK Otomatik Kontrol Ulusal Toplantısı Bildiriler Kitabı*, pages 188–193, Denizli, Turkey, September 2015. In Turkish.

[53] G. Gülmez. A GUI for a software to design decentralized controllers for time-delay systems and control of a coupled tank system. Technical report, Anadolu University, Eskisehir, 2016.

[54] G. Gülmez, S. M. Özer, and A. İftar. A GUI for a software to design decentralized controllers for time-delay systems. In *TOK Otomatik Kontrol Ulusal Toplantısı Bildiriler Kitabı*, Eskisehir, Turkey, September 2016. In Turkish, To Appear.

[55] H. E. Erol and A. İftar. Decentralized controller design by continuous pole placement for commensurate-time-delay systems. In *Proceedings of the 19th IFAC World Congress*, pages 9419–9424, Cape Town, South Africa, August 2014.